Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 24, 2015

A. Bierman
YumaWorks, Inc.
October 21, 2014

NETCONF Efficiency Extensions
draft-bierman-netconf-efficiency-extensions-02

Abstract

This document describes protocol extensions to improve the efficiency of the Network Configuration Protocol (NETCONF). Protocol capabilities and operations are defined to reduce network usage and transaction complexity.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

There is a need for standard mechanisms to allow NETCONF [RFC6241] application designers to manage NETCONF servers more efficiently when used in network environments with poor connectivity, low bandwidth, and/or high latency. In such conditions, it is desirable to minimize network usage wrt/ the size of protocol messages and the number of protocol operations required to perform a network management function.

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

1.1.1. NETCONF

The following terms are defined in [RFC6241]:

- candidate configuration datastore
- client
- configuration data
- datastore
- configuration datastore
- protocol operation
- running configuration datastore
- server
- startup configuration datastore

1.1.2. YANG

The following terms are defined in [RFC6020]:

- container
- data node

- o key leaf
- o leaf
- o leaf-list
- o list

1.1.3. RESTCONF

The following terms are defined in [RESTCONF]:

- o data resource
- o datastore resource

1.1.4. YANG Patch

The following term is defined in [YANG-Patch]:

- o YANG Patch

1.1.5. Terms

The following terms are defined:

- o config ID: An opaque string identifier that represents the state of the running datastore contents on the server. A new config ID is chosen by the server each time the server running configuration datastore is altered in any way.
- o depth filter: A mechanism implemented within the NETCONF server to allow a client to retrieve only a limited number of levels within the a subtree, instead of retrieving the entire subtree.
- o time filter: A mechanism implemented within the NETCONF server to allow a client to retrieve only data that has been modified since a specified data and time.

1.1.6. Tree Diagrams

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

- o Brackets "[" and "]" enclose list keys.

- o Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).

- o Symbols after data node names: "?" means an optional node 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.

1.2. Problem Statement

This document attempts to address the following problems with NETCONF protocol procedures.

1.2.1. Initial Configuration Retrieval

A client application often needs to retrieve the entire running configuration datastore contents, usually at the start of an editing session. The <rpc-reply> for this <get-config> request can be very large (e.g., greater than 250,000 bytes).

If a large number of server connections are lost and then restarted, the quantity of large <rpc-reply> messages from every server could severely impact network performance.

It would be useful if the <hello> message exchange could be enhanced so an entity-tag value for the current running datastore configuration is included in the server <hello> message. A client can cache the server configuration identifier and omit an initial <get-config> operation if the value from the server <hello> message matches the cached value.

1.2.2. Datastore Editing

There are several deficiencies with the NETCONF editing procedures that could be improved.

Multi-operation functions can be required. A single edit can take up to 9 operations. Several operations are required to complete a set of 1 or more edits on a NETCONF server. Each operation uses 1 request and 1 response message. If the candidate datastore is used, then 1 extra operation is required (for the <commit> operation) to activate the edit(s). If the startup datastore is used then 1 extra operation is required (for the <copy-config> operation) to save the running datastore contents in non-volatile storage. If global locking is used, then 2 extra operations are required for each datastore involved (candidate, running, startup) Since the datastore is locked at the start and unlocked at the end of the entire edit operation, these extra roundtrip times are intervals in which the datastore is being locked, but no datastore access is being done.

Obtaining locks can be expensive. If the server has more than 1 datastore (e.g., candidate + running or running + startup), then multiple lock requests are required, since the <lock> and <unlock> operations on affect 1 datastore at a time. This can cause a long delay or even deadlock if multiple clients are attempting to obtain global locks at once. E.g., client 1 holds a lock on the candidate datastore and is trying to lock the running datastore. At the same time, client 2 holds a lock on the running datastore and is trying to lock the candidate datastore.

Using locks can be brittle. NETCONF clients are intended to be programmatic, so is not likely that locks will be long-lived. Global locks are designed to be short-lived since they block write access to the entire datastore. If lock collisions do occur, they are likely to be cleared very quickly. It would be useful if the client could request how long to wait for locks to clear instead of immediately rejecting an edit request due to an 'in-use' error.

Edit operations are implied by <config> content. NETCONF uses a default operation and explicit operation attribute within an arbitrarily complete XML subtree to represent a configuration datastore. There are several corner-cases that are not standardized, and very implementation-dependent:

- interpretation of implied operations vs. explicit operations
- order the edits are processed
- handling of nested operation attributes
- handling of duplicate subtrees
- error handling (code points, number of errors, etc.)
- move operations are not explicit and can interpreted as a request to remove and re-add an entry, not just move user-ordered data

Edit operations are not protected against multi-client alterations. It is a simple and common practice to retrieve a configuration data resource, changing 1 or more fields, and then update the resource on the server. Since retrieval and edit operations are separate there is always a chance that another client has altered the resource after the <get-config> operation, but before the <edit-config> operation, by the first client. Each client could be protected if there was an entity tag associated with each data resource, and an edit request could be rejected if the client attempted to edit a different version of the data resource than expected.

There is no bulk-edit support. If the same edit is needed in multiple instances of a particular data resource, then the data must be repeated for each instance in the <edit-config> or <copy-config> request. The request message size could be minimized if there was a way to apply a set of edits to multiple target nodes at once.

There is no confirmed commit support for the running datastore. The ability to backup the running datastore, change it, and revert it unless the client confirms the changes has nothing to do with the candidate datastore. A NETCONF server with limited memory is not likely to support the candidate datastore. This feature is useful for any type of network-wide configuration change, regardless of device size.

1.2.3. Data Retrieval

NETCONF data retrieval via the <get> and <get-config> operations can be very inefficient. Some vendors do not even support <get> because it can be such a resource-intensive operation and return an enormous amount of data, especially if all server data is requested at once.

A client cannot retrieve just the non-configuration data. The NETCONF <get> operation allows a client to retrieve data from the server but it returns all data, including configuration datastore nodes. The <get-config> operation already returns all configuration datastore nodes.

It was originally thought that <get> should return all nodes so the client would not have to correlate configuration and non-configuration data nodes, since they would be mixed together in the reply. Operational experience has shown that the <get> operation without reasonable filters to reduce the returned data can significantly degrade device performance and return enormous XML instance documents in the <rpc-reply>.

There is no "last-modified" indication or time filtering. The NETCONF protocol has no standard mechanisms to indicate to a client when a datastore was last modified, or to allow a client to retrieve data only if it has been modified since a specified time. This makes polling applications very inefficient because they will regularly burden the server and the network and themselves with retrieval and processing requests for data that has not changed.

There is no simple list instance discovery mechanism. Sometimes the client application wants to discover what data exists on the server, particularly list entries. There is a need for a simple mechanism to retrieve just the key leaf nodes within a subtree. The NETCONF subtree filtering mechanism does provide a very complex way for the client to request just key leafs for specific list entries. A simpler mechanism is needed which will allow the client to discover the list instances present.

There is no subtree depth control. NETCONF filters allow the client to select specific sub-trees within the conceptual datastore on the server. However, sometimes the client does not really need the entire subtree, which may contain many nested list entries, and be very large. There is sometimes a need to limit the depth of the sub-trees retrieved from the server. A consistent and simple algorithm for determining what data nodes start a new level is needed.

The content filter specification is not extensible. The NETCONF <get> and <get-config> operations use a hard-coded content filtering mechanism. They use a "type" XML attribute to indicate which of two filter specification types they support, and a "select" XML attribute if the :xpath capability is supported and an XPath [XPATH] expression filter specification is provided.

This design does not allow additional content filter specification types to be supported by an implementation. It does not allow the standard to be easily extended in a modular fashion. In addition, this design does not allow YANG statements to be used to properly describe the protocol operation. The special "get-filter-element-attributes" YANG extension in the ietf-netconf module is not extensible, and it does not really count as proper YANG, since this extension is outside the YANG language definition.

There is no standard metadata or standard way to retrieve metadata. The <with-defaults> parameter allows 1 specific type of metadata to be returned (i.e., 'report-all-tagged' mode). This ad-hoc approach does not scale well and is not extensible. It would be useful if standard and vendor-specific metadata could be identified and retrieved with standard operations.

1.3. Solution

This document defines some NETCONF protocol operations and new capabilities to reduce network usage and increase functionality at the same time.

All NETCONF efficiency extensions are completely backward-compatible with the current definitions in [RFC6241]. An old client will ignore any new <capability> URIs sent by the server, and will not use the new operations. No existing operations are affected by the new operations, so the extensions will be transparent to an existing

NETCONF client.

1.3.1. Configuration ID Advertisement

A new capability called "config-id" is defined to identify the current running datastore configuration contents with an opaque string. A client can cache this value for each server that supports this capability, along with a copy of its running configuration. When a new session is started, the client can examine the "config-id" <capability> URI sent by the server. If it is the same as the cached value then the client can use the cached running datastore copy instead of sending an initial <get-config> operation to the server. The :config-id capability is ignored in the calculation of the :capability-id capability. Refer to Section 2.1 for details on configuration ID advertisement.

1.3.2. <edit2> Operation

A new NETCONF protocol operation called <edit2> is defined to address the deficiencies described in Section 1.2.2. This operation allows the entire NETCONF edit procedure to be accomplished with 1 request message. The editing procedures are aligned with the resource model defined in [RESTCONF]. Refer to Section 2.2 for details on <edit2> operation.

The "confirmed-commit" procedure has been integrated into the <edit2> operation, and can be supported by any server without requiring support for the candidate datastore. It is optional to implement, based on the "confirmed-edit" capability defined in Section 2.6.

Refer to Section 2.3 for details on the <complete-commit> operation and Section 2.4 for details on the <revert-commit> operation.

1.3.3. <get2> Operation

A new NETCONF protocol operation called <get2> is defined to address the deficiencies described in Section 1.2.3. This operation allows several filter types to be combined to control the data that is returned in the <rpc-reply> message, and an extensible framework for retrieving metadata associated with datastore or data resources. Refer to Section 2.5 for details on <get2> operation.

2. Definitions

This section defines the NETCONF efficiency extensions:

- :config-id Capability
- <edit2> Operation
- <complete-commit> Operation
- <revert-commit> Operation
- <get2> Operation

2.1. "config-id" Capability

2.1.1. Overview

The :config-id capability indicates that the server maintains a config ID for the running configuration datastore. This identifier value is selected by the server and treated as an opaque string by the client.

```
1) Client keeps a cache of server configurations.
2) Server always sends its current config-id value
   in the "config-id" <capability> URI.

     Client                     Server X <hello> message
+---------------+             +-------------------+
|               |             | config-id=1234    |
| Server Config |  <-------   | <capability>      |
|    Cache      |             | ...               |
+---------------+             +-------------------+


3) Client checks cache for server X, config-id=1234.
If found, then OK to use the cached configuration copy.
If not found, then send a <get-config> for the running
configuration to create or update the cached copy.
```

The server SHOULD save the config ID for the running datastore in non-volatile storage. When the server boots or restarts, the initial configuration ID SHOULD be the same as the last instantiation, if the server does not support the :startup capability (so the non-volatile stored version mirrors the running datastore). If the server does support the :startup capability, then the initial configuration ID SHOULD be the same as the version last saved to non-volatile storage.

2.1.1.1. :config-id Capability Example

The :config-id capability is sent in every server <hello> message. The "id" parameter for the :config-id capability is set to the current config ID for the running datastore on the server:

```
# Server starts session 3 with an abbreviated <hello>
<hello xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
 <capabilities>
  <capability>
   urn:ietf:params:netconf:capability:config-id?id=4284
  </capability>
  // ... rest of <capability> elements
 <capabilities>
 <session-id>3</session-id>
</hello>
```

2.1.2. Dependencies

The :config-id capability is not dependent on any other capabilities.

2.1.3. Capability Identifier

The :config-id capability is identified by the following capability string:

urn:ietf:params:netconf:capability:config-id:1.0

This capability MUST be advertised in every server <hello> message. The :config-id capability URI MUST contain an "id" argument assigned an opaque string value indicating the current config ID value for the running datastore. For example:

urn:ietf:params:netconf:capability:config-id:1.0?id=6882391

The current config ID value MUST be updated any time a "netconf-config-change" event would be generated by the server.

If [RFC6470] is supported, then the "config-id" leaf defined in Section 2.6 MUST be included in <netconf-config-change> event notifications.

If the "with-metadata" parameter in the <get2> operation specifies the "config-id" identity, then the server MUST return the current config ID for the running datastore, if the "source" parameter identifies the running datastore. The server MAY maintain config IDs for other datastores as well.

2.1.4. New Operations

The :config-id capability does not introduce any new protocol operations.

2.1.5. Modifications to Existing Operations

The :config-id capability does not modify any existing protocol operations.

2.1.6. Interactions with Other Capabilities

The :config-id capability does not interact with any other capabilities.

2.2. <edit2> Protocol Operation

The <edit2> operation is specified with a YANG "rpc" statement, defined in Section 2.6. This operation allows the entire NETCONF transaction procedure to be performed in a single operation or multiple operations, depending on the input parameters used.

There are no XML attributes used (e.g., "operation" from RFC 6241, "insert", "value" from RFC 6020). Instead, configuration edits are specified with an edit list, using the YANG Patch mechanism defined in [RESTCONF]. This is used instead of a complete XML instance document, e.g. <config> element, to represent an unordered patch list inferred from the diffs. (Although YANG Patch can be used in this mode if client wants to merge or replace the entire configuration datastore).

2.2.1. <edit2> Input

- o target: name of the configuration datastore being edited
- o target-resource: XPath node-set expression representing 1 or more target resources within the datastore to edit.
- o yang-patch: container of ordered edits to apply to the target resource(s).
- o test-only: flag to request that the edit request be validated but no edits should actually be applied
- o if-match: if the entity tag for the target resource(s) does not exactly match the supplied value then the edit request is rejected.
- o with-locking: if present then the server will provide exclusive write access to this <edit2> operation and possible confirmed-commit procedure.

- o max-lock-wait: amount of time the client is willing to wait for locks to clear, if "with-locking" parameter is present.

- o activate-now: if present and the target is the candidate datastore, then an implicit <commit> operation will be performed if the edit operation is successfully applied.

- o nvstore-now: if present and the server supports the startup datastore, and the edits have been activated in the running datastore, then an implicit <copy-config> operation (from the running to the startup datastore) will be attempted by the server.

- o confirmed: request that a confirmed commit be started or extended.

- o confirm-timeout: the amount of time for the server to wait for an <edit2> request that extends, a <complete-commit> request to finish, or a <revert-commit> request to cancel a confirmed commit procedure in progress.

- o persist: identifier string to use in the "persist-id" parameter to extend, complete, or cancel a confirmed commit procedure.

- o persist-id: identifier string to extend a confirmed commit procedure in progress.

2.2.2. <edit2> Output

Positive Response:

This operation returns data containing a "yang-patch-status" report (defined in [RESTCONF]) instead of an "ok" element. This report contains an "ok" element that is present if the entire operation succeeded.

Error Response:

The <rpc-error> element can be returned, e.g., if the message contains invalid parameter syntax. The server MUST report editing errors in the "edit" list within the "yang-patch-status" container.

2.2.3. <edit2> YANG Tree Diagram

```
Key: DRI = data-resource-identifier


+---x edit2
    +--ro input
    |  +--ro target
```

```
         |  |  +--ro (datastore-target)
         |  |     +--:(candidate)
         |  |     |  +--ro candidate?   empty
         |  |     +--:(running)
         |  |        +--ro running?     empty
         |  +--ro target-resource?   yang:xpath1.0
         |  +--ro yang-patch
         |  |  +--ro patch-id?   string
         |  |  +--ro comment?    string
         |  |  +--ro edit [edit-id]
         |  |     +--ro edit-id      string
         |  |     +--ro operation    enumeration
         |  |     +--ro target       data-resource-identifier
         |  |     +--ro point?       data-resource-identifier
         |  |     +--ro where?       enumeration
         |  |     +--ro value
         |  +--ro test-only?         empty
         |  +--ro if-match?          yang-entity-tag
         |  +--ro with-locking?      empty
         |  +--ro max-lock-wait?     uint32
         |  +--ro activate-now?      empty
         |  +--ro nvstore-now?       empty
         |  +--ro confirmed?         empty
         |  +--ro confirm-timeout?   uint32
         |  +--ro persist?           string
         |  +--ro persist-id?        string
         +--ro output
            +--ro yang-patch-status
               +--ro patch-id?        string
               +--ro (global-status)?
               |  +--:(global-errors)
               |  |  +--ro errors
               |  |        +--ro error
               |  |        +--ro error-type       enumeration
               |  |        +--ro error-tag        string
               |  |        +--ro error-app-tag?   string
               |  |        +--ro error-path?      DRI
               |  |        +--ro error-message?   string
               |  |        +--ro error-info
               |  |
               |  +--:(ok)
               |     +--ro ok?              empty
               +--ro edit-status
                  +--ro edit [edit-id]
                     +--ro edit-id  string
                     +--ro (edit-status-choice)?
                        +--:(ok)
                        |  +--ro ok?         empty
```

```
                        +--:(errors)
                           +--ro errors
                              +--ro error
                                 +--ro error-type       enumeration
                                 +--ro error-tag        string
                                 +--ro error-app-tag?   string
                                 +--ro error-path?      DRI
                                 +--ro error-message?   string
                                 +--ro error-info
```

2.2.4. <edit2> Example

In this example, an "all-in-one" YANG Patch edit is shown. the following conditions apply:

- The server supports the :candidate and :startup capabilities
- The "example-ex" YANG module is supported by the server

The starting state of the "/forests" data structure is described in Appendix B.2. The client is adding an "oak" tree and changing the location of the "birch" tree in the "north" forest.

```
<rpc message-id="101"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ex="http://example.com/ns/example-ex">
    <target><candidate/></target>
    <target-resource>
      /ex:forests/ex:forest[ex:name='north']
    </target-resource>
    <yang-patch>
      <patch-id>north-forest-patch</patch-id>
      <comment>
        Add an oak tree and change location of the birch tree
      </comment>
      <edit>
        <edit-id>oak</edit-id>
        <operation>create</operation>
        <target>/ex:trees</target>
        <value>
          <ex:tree>
            <ex:name>oak</ex:name>
            <ex:location>hillside</ex:location>
          </ex:tree>
        </value>
      </edit>
      <edit>
        <edit-id>birch</edit-id>
        <operation>merge</operation>
        <target>/ex:trees/ex:tree/birch</target>
        <value>
          <ex:location>west valley</ex:location>
        </value>
      </edit>
    </yang-patch>
    <activate-now/>
    <nvstore-now/>
  </edit2>
</rpc>
```

The edit succeeds, and the "yang-patch-status" container is returned to the client with the <ok/> status for both tree edits:

```
<rpc-reply message-id="101"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <yang-patch-status
    xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ex="http://example.com/ns/example-ex">
    <patch-id>north-forest patch</patch-id>
    <ok/>
    <edit-status>
      <edit>
        <edit-id>oak</edit-id>
        <ok/>
      </edit>
      <edit>
        <edit-id>birch</edit-id>
        <ok/>
      </edit>
    </edit-status>
  </yang-patch-status>
</rpc-reply>
```

Refer to Appendix B.3 for additional <edit2> protocol operation examples.

2.3. <complete-commit> Operation

A new NETCONF protocol operation called <complete-commit> is defined to complete a confirmed commit procedure.

2.3.1. <complete-commit> Input

There is one optional parameter for this protocol operation:

- o persist-id: an identifier string that MUST match the "persist" value, if it was used in the confirmed-commit procedure.

2.3.2. <complete-commit> Output

Positive Response:

The there is a confirmed-commit procedure in progress and it is successfully completed, then an <ok/> element is returned.

Negative Response: An <rpc-error> response is sent if the request cannot be completed for any reason.

2.3.3. <complete-commit> YANG Tree Diagram

```
+---x complete-commit
   +--ro input
      +--ro persist-id?   string
```

2.3.4. <complete-commit> Example

In this example, the client has previously started a confirmed commit procedure using the "persist" parameter set to the value "abcdef".

```
<rpc message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <complete-commit
    xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <persist-id>abcdef</persist-id>
  </complete-commit>
</rpc>


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

2.4. <revert-commit> Operation

A new NETCONF protocol operation called <revert-commit> is defined to cancel a confirmed commit procedure and revert the running datastore. The <cancel-commit> operation in [RFC6241] cannot be used because it requires the implementation of the candidate capability.

2.4.1. <revert-commit> Input

There is one optional parameter for this protocol operation:

- o persist-id: an identifier string that MUST match the "persist" value, if it was used in the confirmed-commit procedure.

2.4.2. <revert-commit> Output

Positive Response:

If there is a confirmed-commit procedure in progress and it is successfully cancelled, and the running datastore successfully reverted, then an <ok/> element is returned.

Negative Response: An <rpc-error> response is sent if the request cannot be completed for any reason.

2.4.3. <revert-commit> YANG Tree Diagram

```
+---x revert-commit
   +--ro input
      +--ro persist-id?   string
```

2.4.4. <revert-commit> Example

In this example, the client has previously started a confirmed commit procedure using the "persist" parameter set to the value "abcdef".

```
<rpc message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <revert-commit
    xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <persist-id>abcdef</persist-id>
  </revert-commit>
</rpc>
```

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

2.5. <get2> Protocol Operation

The <get2> operation is specified with a YANG "rpc" statement, defined in Section 2.6. A specific datastore is selected for the source of the retrieval operation. Several different types of filters are provided. Filters are combined in a conceptual "logical-AND" operation, and are optional to use by the client. Not all filtering mechanisms are mandatory-to-implement for the server.

2.5.1. Depth Filters

A depth filter indicates how many subtree levels should be returned in the <rpc-reply>. This filter is specified with the "depth" input parameter for the <get2> protocol operation. The default "0" indicates that all levels from the requested subtrees should be returned.

A new level is started for each YANG data node within the requested subtree. All top level data nodes are considered to be child nodes (level 1) of a conceptual <config> root.

If no content filters are provided, then level 1 is considered to include all top-level data nodes within the source datastore. Otherwise only the levels in selected subtrees will be considered, and not any additional top-level data nodes.

If the depth requested is equal to "1", then only the requested data nodes (or top-level data nodes) will be returned. This mechanism can be used to detect the existence of containers and list entries within a particular subtree, without returning any of the descendant nodes.

Higher depth values indicates the number of descendant nodes to include in the response. For example, if the depth requested is equal to "2", then only the requested data nodes (or top-level data nodes) and their immediate child data nodes will be returned.

2.5.2. Time Filters

A time filter specifies that data should only be returned if the last-modified timestamp for the target datastore is more recent than the timestamp specified in the "if-modified-since" parameter.

If this feature is supported, then the server will maintain a "last-modified" timestamp for the running datastore. The server MAY support additional nested timestamps for data nodes within the datastore. The server MAY support timestamps for other datastores.

When a request containing the "if-modified-since" parameter is received, the server will compare that timestamp to the "last-modified" timestamp for the source datastore. If it is greater than the specified value then data may be returned (depending on other filters). If the datastore timestamp value is less than or equal to the specified value, then an empty <data> element will be returned in the <rpc-reply>.

If the "full-delta" parameter is present, and the server maintains "last-modified" timestamps for any data nodes within the source datastore, then the same type of comparison will be done for the data node to determine if it should be included in the response. If no "last-modified" timestamp is maintained for a data node, then the server will use the "last-modified" timestamp for its nearest ancestor, or for the datastore itself if there are none.

2.5.3. <get2> Input

o source: A container indicating the conceptual datastore for the retrieval request.

- filter-spec: A choice indicating the content filter specification for the retrieval request.

- keys-only: A leaf indicating that only the key leafs, combined with other filtering criteria, should be returned.

- if-modified-since: A leaf indicating the time filter specification for the retrieval request, according to the procedures in Section 2.5.2.

- full-delta: If present and the "if-modified-since" parameter is also present, then the entire datastore will be filtered by last modification time, not just the entire datastore.

- depth: A leaf indicating the subtree depth level for the retrieval request, according to the procedures in Section 2.5.1.

- with-defaults: A leaf indicating the type of defaults handling requested, according to procedures in [RFC6243].

- with-metadata: A leaf-list indicating the specific metadata that the server should add to the response, such as "last-modified" or "etag", encoded in XML according to the schema in Section 2.7.

- with-locking: if present then the server will provide exclusive write access to this <get2> operation so the target datastore is not modified during the entire retrieval operation.

- max-lock-wait: amount of time the client is willing to wait for locks to clear, if "with-locking" parameter is present.

2.5.4. <get2> Output

Positive Response: A <data> element is returned which contains the data corresponding to the input parameters specified in the request. The child nodes of the <data> container correspond to top-level YANG data nodes.

If the server supports the "timestamps" YANG feature, and the target is the running datastore, then a "last-modified" attribute SHOULD be included in the <rpc-reply> element.

Negative Response: An <rpc-error> response is sent if the request cannot be completed for any reason.

2.5.5. <get2> YANG Tree Diagram

```
      +---x get2
         +--ro input
         |  +--ro source
         |  |  +--ro (datastore-source)?
         |  |     +--:(candidate)
         |  |     |  +--ro candidate?   empty
         |  |     +--:(running)
         |  |     |  +--ro running?     empty
         |  |     +--:(startup)
         |  |     |  +--ro startup?     empty
         |  |     +--:(url)
         |  |     |  +--ro url?         inet:uri
         |  |     +--:(operational)
         |  |        +--ro operational? empty
         |  +--ro (filter-spec)?
         |  |  +--:(subtree-filter)
         |  |  |  +--ro subtree-filter
         |  |  +--:(xpath-filter)
         |  |     +--ro xpath-filter?        yang:xpath1.0
         |  +--ro keys-only?           empty
         |  +--ro if-modified-since?   yang:date-and-time
         |  +--ro full-delta?          empty
         |  +--ro depth?               uint32
         |  +--ro with-defaults?       with-defaults-mode
         |  +--ro with-metadata*       identityref
         |  +--ro with-locking?        empty
         |  +--ro max-lock-wait?       uint32
         +--ro output
            +--ro data
```

2.5.6. <get2> Example

In this example, the retrieval the "forests" resource is shown. the following conditions apply:

- The server supports the :candidate and :startup capabilities
- The "example-ex" YANG module is supported by the server

The starting state of the "/forests" data structure is described in Appendix B.2. The client is retrieving just the "forests" node, along with the "last-modified" and "etag" metadata for that node. The "config-id" for the datastore is also requested. Locking is requested (with a maximum lock wait time of 5 seconds), just to make sure the metadata does not change during the request.

```
<rpc message-id="104"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ncex="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <subtree-filter>
      <forests xmlns="http://example.com/ns/example-ex" />
    </subtree-filter>
    <depth>1</depth>
    <with-metadata>ncex:timestamps</with-metadata>
    <with-metadata>ncex:etags</with-metadata>
    <with-metadata>ncex:config-id</with-metadata>
    <with-locking />
    <max-lock-wait>5</max-lock-wait>
  </get2>
</rpc>
```

The server has a "forests" node so this node is returned along with the requested metadata for the node. Note that the XML namespace for the "ncex" metadata is the XSD target namespace defined in Section 2.7, not the YANG namespace URI defined in Section 2.6.

```
<rpc-reply message-id="104"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
        xmlns:m="urn:ietf:params:xml:ns:netconf:netconf-ex:1.0"
        m:last-modified="2012-09-09T02:00:00Z"
        m:config-id="3aee5601">
    <forests xmlns="http://example.com/ns/example-ex"
             m:last-modified="2012-09-09T02:00:00Z"
             m:etag="3aee5601" />
  </data>
</rpc-reply>
```

Refer to Appendix B.4 for additional <get2> protocol operation examples.

2.6. NETCONF-EX YANG Module

This module imports the "with-defaults-parameters" grouping from [RFC6243].

Several YANG features are imported from [RFC6241]. These correspond to the NETCONF capabilities (e.g., candidate, url, startup, xpath) but defined as YANG features instead of URIs.

Some data types are imported from [RFC6991]:

- date-and-time
- uri
- xpath1.0

Two YANG groupings are imported from [YANG-Patch]:

- yang-patch
- yang-patch-status

One notification is augmented from [RFC6470].

- netconf-configuration-change

RFC Ed.: update the date below with the date of RFC publication and remove this note.

<CODE BEGINS> file "ietf-netconf-ex@2014-10-21.yang"

```
module ietf-netconf-ex {

  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-ex";
  prefix ncex;

  import ietf-inet-types {
    prefix inet;
  }

  import ietf-netconf {
    prefix nc;
  }

  import ietf-netconf-notifications {
    prefix ncn;
  }

  import ietf-netconf-with-defaults {
    prefix ncwd;
  }

  import ietf-yang-patch {
    prefix yp;
  }

  import ietf-yang-types {
    prefix yang;
  }

  organization
```

```
        "IETF NETCONF (Network Configuration Protocol) 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: Bert Wijnen
                   <mailto:bertietf@bwijnen.net>

         Editor:   Andy Bierman
                   <mailto:andy@yumaworks.com>";

      description
        "This module contains a collection of YANG definitions for the
         efficient operation of a NETCONF server. Protocol operations
         are defined to reduce network usage and transaction complexity.

         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.";

      // RFC Ed.: replace XXXX with actual RFC number and remove this
      // note.

      // RFC Ed.: remove this note
      // Note: extracted from
      // draft-bierman-netconf-efficiency-extensions-02.txt

      // RFC Ed.: update the date below with the date of RFC publication
      // and remove this note.
      revision "2014-10-21" {
        description
          "Initial revision. <get2> operation originally published
           in draft-bierman-netconf-get2-03.txt";
        reference
          "RFC XXXX: NETCONF Efficiency Extensions";
```

```
     }


     /* Features */

     feature timestamps {
       description
         "This feature indicates that the server implements
          the <get2> operations parameters which require
          last modification timestamps to be maintained by
          the server.

          If this feature is advertised then one global
          'last-modified' timestamp for the entire
          running configuration datastore MUST be supported.

          The server MAY support additional timestamps
          for additional datastores and data nodes
          within a datastore.  The 'with-metadata'
          parameter can be used to identify
          which data nodes support a 'last-modified'
          timestamp.";
     }

     feature with-defaults {
       description
         "This feature indicates that the server supports the
          'with-defaults' parameter for the <get2> operation.
          A NETCONF server SHOULD support this feature.";
       reference
         "RFC 6243: With-defaults Capability for NETCONF";
     }

     feature confirmed-edit {
       description
         "This feature indicates that the server supports the
          confirmed commit procedure for the <edit2> protocol
          operation.";
     }


     /* Identities */

     identity metadata {
       description
         "Base for all metadata identifiers used by the
          'with-metadata' parameter in the <get2> operation.";
     }
```

```
     identity timestamps {
       base metadata;
       description
          "Describes metadata identifying the last modification
           time of the associated datastore or data resource.";
     }

     identity etags {
       base metadata;
       description
          "Describes metadata identifying the entity tag value
           of the associated datastore or data resource.";
     }

     identity config-id {
       base metadata;
       description
          "Describes metadata identifying the config ID
           of the associated datastore or data resource.";
     }


     /* Typedefs */

     typedef yang-entity-tag {
       type string;
       description
         "Contains an opaque string representing a specific instance
          of a datastore or data resource. A client can use this
          string for equality comparisons between yang-entity-tag
          values.

          If any configuration data node values changes, or the
          relative order of any user-ordered data changes, then
          the server MUST change the entity tag value for the
          running datastore to a different value.  If the server
          maintains entity-tag values for configuration data nodes,
          then the server MUST change the yang-entity-tag value for
          any affected data node.

          Only yang-entity-tag values for the same target resource
          instance can be compared. Only the 'strong entity tag'
          form is required.  A server MAY support the 'weak
          entity tag' form.  If so, then 2 YANG data node resource
          instances are considered to be equivalent if they
          contain the same value subtrees and all user-ordered
          data nodes share the same relative order.";
       reference
```

```
        "RFC 2616, section 3.11.";
   }


   /* Groupings */

   grouping lock-parms {
     description
       "Common parameters to control datastore locking.";

     leaf with-locking {
       type empty;
       description
        "If this parameter is present then the request MUST be
         performed with exclusive write access to all datastores
         involved in the operation.  An 'operation-not-supported'
         error-tag value is returned if the target datastore for
         the operation does not support locking (e.g., 'url' or
         'operational').

         If the server cannot provide exclusive write access
         for the entire requested operation then an 'in-use'
         error-tag value is returned.

         If the 'max-lock-wait' parameter is also present then
         the server MAY choose to wait up to that amount of
         time attempting to obtain exclusive write access,
         before returning an error.";
     }

     leaf max-lock-wait {
       when "../with-locking" {
         description
           "Only relevant if locking is requested.";
       }
       type uint32 {
         range "1 .. 600";
       }
       units seconds;
       description
        "If this parameter is present and the 'with-locking'
         parameter is also present, then the server MAY wait
         up to the specified number of seconds attempting
         to obtain exclusive write access for the requested
         operation.";
     }
   }
```

```
     /* Protocol Operations */

     rpc get2 {
       description
         "Retrieve NETCONF datastore information";
       input {
         container source {
           description
             "The datastore (or non-configuration data)
              to use for the source for the retrieval operation.";

           choice datastore-source {
             default running;
             description
               "The configuration source for the retrieval operation.
                The running configuration is the default choice if
                this parameter is not present.";
             leaf candidate {
               if-feature nc:candidate;
               type empty;
               description
                 "The candidate configuration datastore is the
                  retrieval source.";
             }
             leaf running {
               type empty;
               description
                 "The running configuration datastore is the
                  retrieval source.";
             }
             leaf startup {
               if-feature nc:startup;
               type empty;
               description
                 "The startup configuration datastore is the
                  retrieval source.";
             }
             leaf url {
               if-feature nc:url;
               type inet:uri;
               description
                 "The URL-based configuration is the
                  retrieval source.";
             }
             leaf operational {
               type empty;
               description
                "The retrieval source is the collection of all
```

```
                 operational (non-configuration) data nodes supported
                 by the server.

                 Any ancestor container and/or list and list key nodes
                 are also returned. No other leafs or leaf-lists will
                 be included in the reply.

                 The server MAY return ancestor container, and/or list
                 and list key nodes that do not contain any
                 non-configuration nodes. This can occur for several
                 reasons, e.g., the implementation streams replies
                 and cannot defer instrumentation or access control
                 filtering of descendant data nodes.";
             }
           }
         }

         choice filter-spec {
           description
             "The content filter specification for this request";

           anyxml subtree-filter {
             description
               "This parameter identifies the portions of the
                target datastore to retrieve.";
             reference "RFC 6241, Section 6.";
           }
           leaf xpath-filter {
             if-feature nc:xpath;
             type yang:xpath1.0;
             description
               "This parameter contains an XPath expression
                identifying the portions of the target
                datastore to retrieve.";
           }
         }

         leaf keys-only {
           type empty;
           description
             "This parameter selects only data nodes which
              are key leaf nodes.  Parent container and
              list nodes are also returned, but no other leafs,
              or any leaf-lists will be included in the reply.";
         }

         leaf if-modified-since {
           if-feature timestamps;
```

```
          type yang:date-and-time;
          description
            "This parameter selects the target datastore
             only if the last-modified timestamp for the
             datastore is more recent than the specified time.
             If not, then an empty <data> element is returned.

             If the target datastore does not maintain a
             last-modified timestamp, then this parameter is
             ignored.";
        }

        leaf full-delta {
          if-feature timestamps;
          type empty;
          description
            "This parameter selects only data nodes which
             have been modified since the specified time.
             It is ignored unless the 'if-modified-since'
             parameter is also provided and the target datastore
             supports a last-modified timestamp.";
        }

        leaf depth {
          type uint32;
          default 0;
          description
            "This parameter selects how many conceptual
             sub-tree levels should be returned in the
             <rpc-reply>.

             If this parameter is equal to '0', then entire
             subtrees will be returned.

             If this parameter is greater than '0', then
             only the specified number of subtree levels will
             be returned.";
        }

        uses ncwd:with-defaults-parameters {
          if-feature with-defaults;
          description
            "This parameter controls the retrieval of
             default values.";
           reference
             "RFC 6243: With-defaults Capability for NETCONF";
        }
```

```
          leaf-list with-metadata {
            type identityref {
              base metadata;
            }
            description
              "This parameter will cause the server to return
               metadata in the <rpc-reply> (e.g. as XML attributes
               in XML encoding) associated with the specified
               metadata identity. If the server does not support
               any specified metadata identifier, then the
               operation fails with an 'invalid-value' error.";
          }

          uses lock-parms {
            description
              "Exclusive write access can be requested to
               ensure that no other sessions modify the
               configuration data during the retrieval operation";
          }

        }

        output {
          anyxml data {
            description
              "Copy of the requested datastore subset which
               matched the filter criteria (if any).
               An empty data container indicates that the
               request did not produce any results.";
          }
        }
      }


      rpc edit2 {
        description
          "Edit NETCONF datastore contents.
           All operations requested in the yang-patch edit list
           are applied, or the target datastore is left unchanged.";

        input {
          container target {
            description
              "The datastore to use as the target for this
               edit operation.";

            choice datastore-target {
              mandatory true;
```

```
            description
              "The configuration target for the edit operation.";

            leaf candidate {
              if-feature nc:candidate;
              type empty;
              description
                "The candidate configuration datastore is the
                 edit target.";
            }
            leaf running {
              if-feature nc:writable-running;
              type empty;
              description
                "The running configuration datastore is the
                 edit target.";
            }
          }
        }

        leaf target-resource {
          if-feature nc:xpath;
          type yang:xpath1.0;
          description
            "This parameter identifies 1 or more data node
             instances for which the yang-patch edits
             will be applied. The target-resource expression
             MUST evaluate to a node-set result.

             Each operation in the yang-patch edit list will
             be applied to each target-resource instance, as if
             it were the document root for the operation.

             If multiple instances are represented by the
             target-resource value, then the server will apply
             all edits to all instances.  If any errors occur,
             then all edits from this request will be undone
             from the target datastore.

             The user MUST have appropriate write permissions for
             all data accessed by every operation within the edit
             list.

             If this parameter is not present or not supported
             then the target resource is the root node of the
             datastore identified by the 'target' parameter.";
        }
```

```
         uses yp:yang-patch {
           description
             "The yang-patch parameter contains the ordered list
              of edits to perform on the target resource(s).

              The conceptual document root for the 'target'
              parameter is defined to be the value of a data node
              represented by the 'target-resource' parameter or the
              target datastore conceptual root node if that parameter
              is not present.";
         }

         leaf test-only {
           type empty;
           description
             "If this parameter is present the server will not
              actually perform the requested edits. Instead the
              edit request will be validated as if it were going
              to be applied.  Any parameter errors or datastore
              validation errors SHOULD be reported in the response.

              No attempt to apply, activate the edits or save them
              in non-volatile storage will be made if this parameter
              is present.";
         }

         leaf if-match {
           type yang-entity-tag;
           description
             "If this parameter is set, then the entire edit request
              will be rejected unless the entity tag for the target
              resource matches this value. An rpc-error with
              an 'operation-failed' error-tag value MUST returned,
              and the edit operation MUST NOT be attempted.
              The 'error-app-tag' field SHOULD be set to
              'precondition-failed'.

              If the target datastore does not maintain a
              last-modified timestamp, then this parameter is
              ignored.";
         }

         uses lock-parms {
           description
             "Exclusive write access can be requested to
              ensure that no other sessions modify the
              configuration data during the edit operation
              and possibly the entire confirmed commit procedure.
```

```
                 If the 'with-locking' parameter is used to start or
                 extend a confirmed commit procedure, then the
                 exclusive write access will be maintained until the
                 confirmed commit procedure terminates somehow.

                 If the 'with-locking' parameter is used for a
                 plain edit operation, then exclusive write access
                 will be maintained until this operation has completed.";
           }

           leaf activate-now {
             type empty;
             description
               "If present and the edit operation succeeds,
                then the server will activate the configuration
                changes right away. The server will conceptually
                perform a <commit> operation after the edit
                operation. The user MUST have execute
                permission for the <commit> operation or
                the operation fails with an 'access-denied' error.

                This parameter has no affect unless the
                'datasource-target' choice is the 'candidate' leaf.";
           }

           leaf nvstore-now {
             type empty;
             description
               "If present and the edit operation succeeds,
                and the configuration changes are activated
                in the running datastore, then the server
                will persist the configuration changes right away
                in non-volatile store. The server will conceptually
                perform a <copy-config> operation from the running
                to the startup datastore. The user MUST have execute
                permission for the <copy-config> operation or
                the operation fails with an 'access-denied' error.

                This parameter has no affect unless the
                'startup' capability is supported by the server.";
           }

           leaf confirmed {
             if-feature confirmed-edit;
             type empty;
             description
               "If the requested edit operation succeeds and the
                configuration changes are applied to the running
```

```
             datastore, then a confirmed commit procedure is
             requested by the client.

             A confirmed commit procedure is an <edit2> operation
             that contains this parameter.  The <complete-commit>
             operation is used to complete the confirmed commit
             procedure. The <revert-commit> operation is used
             to cancel the confirmed commit procedure and
             revert the running datastore back to the contents
             before the first confirmed commit operation.

             If no <complete-commit> or <revert-commit> operation
             is invoked within the timeout interval then
             the server will revert the running datastore
             back to the contents before the first confirmed
             edit operation.

             This is the same as the confirmed commit procedure
             in RFC 6241 except the candidate capability is
             not required.

             The server will save the running datastore contents
             before the edit operation is activated, if there
             is no confirmed edit already in progress.

             If the 'with-locking' parameter is present then
             the server will maintain exclusive write access
             for the specified session until the confirmed
             edit procedure is completed somehow.";
          reference
            "RFC 6241, Section 8.3.4.1";
        }

        leaf confirm-timeout {
          when "../confirmed" {
            description
              "Only relevant if the <confirmed>parameter is present";
          }
          if-feature confirmed-edit;
          type uint32 {
            range "1..max";
          }
          units "seconds";
          default "600";   // 10 minutes
          description
            "The timeout interval for a confirmed edit procedure.
             If exclusive write access was granted for this confirmed
             commit procedure, then it is removed if the timeout
```

```
              occurs and the confirmed commit procedure is terminated.";

           reference "RFC 6241, Section 8.3.4.1";
         }

         leaf persist {
           if-feature confirmed-edit;
           type string;
           description
             "This parameter is used to make a confirmed commit
              procedure persistent.  A persistent confirmed commit
              is not aborted if the NETCONF session terminates.
              The only way to abort a persistent confirmed commit
              is to let the timer expire,  or to use the
              <revert-commit> operation.

              The value of this parameter is a token that MUST be
              given in the 'persist-id' parameter of the <edit2>,
              <complete-commit>, or <revert-commit> operations in
              order to extend, confirm, or cancel the persistent
              confirmed commit procedure.

              The token SHOULD be a random string.";
           reference "RFC 6241, Section 8.3.4.1";
         }

         leaf persist-id {
           if-feature confirmed-edit;
           type string;
           description
             "This parameter is given in order to extend a persistent
              confirmed edit.  The value must be equal to the value
              given in the 'persist' parameter to the <commit>
              operation. If it does not match, the operation fails
              with an 'invalid-value' error.";
             reference "RFC 6241, Section 8.3.4.1";
         }
       }

       output {
         uses yp:yang-patch-status;
       }
     }


     rpc complete-commit {
       if-feature confirmed-edit;
       description
```

```
         "This operation is used to complete an ongoing confirmed
          commit procedure. If exclusive write access was granted
          for this confirmed commit procedure, then it is removed
          if this operation is successfully completed.

          If the confirmed commit is persistent, the parameter
          'persist-id' MUST be given, and it MUST match the value of
          the 'persist' parameter given in the <edit2> operation.
          If not confirmed commit procedure is in progress then
          the operation fails with an 'operation-failed' error.";
       reference "RFC 6241, Section 8.4.5.1";

       input {
         leaf persist-id {
           type string;
           description
             "This parameter is given in order to complete a
              persistent confirmed commit procedure. The
              value MUST be equal to the value given in the
              'persist' parameter to the <edit2>  operation.
              If it does not match, the operation fails with
              an 'invalid-value' error.";
         }
       }
     }


     rpc revert-commit {
       if-feature confirmed-edit;
       description
         "This operation is used to cancel an ongoing confirmed commit.
          If exclusive write access was granted for this confirmed
          commit procedure, then it is removed if this operation
          is successfully completed.

          If the confirmed commit is persistent, the parameter
          'persist-id' MUST be given, and it MUST match the value of
          the 'persist' parameter.  If not confirmed commit
          procedure is in progress then the operation fails
          with an 'operation-failed' error.";
       reference "RFC 6241, Section 8.4.4.1";

       input {
         leaf persist-id {
           type string;
           description
             "This parameter is given in order to cancel a persistent
              confirmed commit and revert the running configuration
```

```
              datastore to its state before the confirmed commit
              procedure started.  The value MUST be equal to the value
              given in the 'persist' parameter to the <edit2>
              operation.

              If it does not match, the operation fails with an
             'invalid-value' error.";
         }
       }
     }


     /* Notifications */

     augment /ncn:netconf-config-change {
       description
         "Add the updated config-id capability value
          to a configuration change event.";

       leaf config-id {
         type string;
         description
           "Contains the new configuration ID for the
            running datastore on the server, representing
            the datastore after the configuration changes.";
       }
     }



   }

   <CODE ENDS>
```

2.7. XSD for NETCONF-EX Metadata

The following XML Schema document [XSD] defines the "last-modified" and "etag" attributes, described within this document. The "last-modified" attribute is only relevant if the server supports the "timestamps" YANG feature within the "ietf-netconf-ex" YANG module.

The "last-modified" attribute uses the XSD data type "dateTime", in accordance with Section 3.2.7.1 of XML Schema Part 2: Datatypes. This is equivalent to the YANG data type "date-and-time".

The "etag" attribute uses the XSD data type "string", in accordance with the "yang-entity-tag" YANG typedef defined in Section 2.6.

The "config-id" attribute uses the XSD data type "string".

<CODE BEGINS> file "netconf-ex.xsd"

```
 <?xml version="1.0" encoding="UTF-8"?>
 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns="urn:ietf:params:xml:ns:netconf:netconf-ex:1.0"
   targetNamespace="urn:ietf:params:xml:ns:netconf:netconf-ex:1.0"
   elementFormDefault="qualified"
   attributeFormDefault="unqualified"
   xml:lang="en">

  <xs:annotation>
    <xs:documentation>
      This schema defines the syntax for the "last-modified"
      and "etag" attributes described within this document.
    </xs:documentation>
  </xs:annotation>

  <!--
    config-id attribute
    -->
  <xs:attribute name="config-id" type="xs:string">
    <xs:annotation>
      <xs:documentation>
        This attribute indicates the current config ID
        for the running configuration datastore,
        corresponding to the XML element containing this attribute.
      </xs:documentation>
    </xs:annotation>
  </xs:attribute>

  <!--
    last-modified attribute
    -->
  <xs:attribute name="last-modified" type="xs:dateTime">
    <xs:annotation>
      <xs:documentation>
        This attribute indicates the date and time when
        a modification was last detected by the server
        for the datastore or data node corresponding to
        the XML element containing this attribute.
      </xs:documentation>
    </xs:annotation>
  </xs:attribute>

  <!--
    etag attribute
```

```
       -->
     <xs:attribute name="etag" type="xs:string">
       <xs:annotation>
         <xs:documentation>
           This attribute indicates the entity tag
           for the datastore or data node corresponding to
           the XML element containing this attribute.
         </xs:documentation>
       </xs:annotation>
     </xs:attribute>

    </xs:schema>

   <CODE ENDS>
```

3. IANA Considerations

3.1. NETCONF-EX XML Namespace

This document registers a URI in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested:

```
URI: urn:ietf:params:xml:ns:netconf:netconf-ex:1.0
Registrant Contact: The NETCONF WG of the IETF.
XML: N/A, the requested URI is an XML namespace.
```

3.2. NETCONF-EX XML Schema

This document registers a URI for the NETCONF XML schema in the IETF XML registry [RFC3688].

```
// RFC Ed. remove this line and uncomment next line when published
//IANA has updated the following URI to reference this document.

URI: urn:ietf:params:xml:schema:netconf-ex
```

3.3. NETCONF-EX YANG Module

This document registers 1 YANG module in the YANG Module Names registry [RFC6020].

```
name:         ietf-netconf-ex
namespace:    urn:ietf:params:xml:ns:yang:ietf-netconf-ex
prefix:       ncex
// RFC Ed. remove this line and replace XXXX in next line
reference:    RFC XXXX
```

4. Security Considerations

This document does not introduce any new security concerns in addition to those specified in [RFC6241], section 9.

5. Change Log

-- RFC Ed.: remove this section before publication.

5.1. 01 to 02

5.1.1. Removed :encoding Capability

The :encoding URI exchange was removed because the developers who want to use JSON are using RESTCONF instead of NETCONF. The NETCONF protocol should support a binary format that can be streamed by servers. This should be done as a separate standards effort.

5.2. 00 to 01

5.2.1. Removed :capability-id Capability

The :capability-id URI exchange was removed because the NETCONF protocol does not allow the server to delay its <hello> message so the client cannot choose the full or abbreviated <hello>.

This makes the :capability-id URI exchange unworkable for several reasons:

- o Since the client cannot select the server hello format based on its own notion of the cached capability set, the server must be configured to always use the full or always use the abbreviated <hello> message.
- o All clients must support the new capability exchange or the server cannot practically be configured to use the abbreviated <hello> message.
- o Since the client will not know the capability-id value for a server the first time the particular value is seen, the "schema" list in the "ietf-netconf-monitoring" YANG module would have to be mandatory-to-implement by both client and server, and mandatory-to-use by the client.
- o Forcing the client to perform a <get> request and wait for an <rpc-reply> before using the NETCONF session introduces 1 round-trip of extra latency into the protocol.
- o Forcing the client to perform a <get> request and wait for an <rpc-reply> before using the NETCONF session introduces extra complexity into the protocol.

5.2.2. RESTCONF Alignment

The YANG module was updated to align with new RESTCONF and YANG Patch drafts. The "location" leaf has been removed from the "yang-patch-status" grouping.

6. Normative References

[RESTCONF]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-02 (work in progress), October 2014.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, June 2011.

[RFC6470] Bierman, A., "Network Configuration Protocol (NETCONF) Base Notifications", RFC 6470, February 2012.

[RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013.

[XPATH] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation REC-xpath-19991116, November 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>.

[XSD] Malhotra, A. and P. Biron, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>.

[YANG-Patch]
Bierman, A., Bjorklund, M., Watsen, K., and R. Fernando, "YANG Patch Media Type", draft-ietf-netconf-yang-patch-01 (work in progress), July 2014.

Appendix A. Open Issues

A.1. resource-identifier-type

The resource-identifier-type typedef from yang-patch is a RESTCONF path expression, not an XPath path expression. The error-path parameter also uses RESTCONF path strings. Should either or both of these be XPath instead?

A.2. no YANG for top-level message nodes

The YANG module of the node is needed for JSON encoding, but there is no YANG schema definition for the <rpc>, <rpc-reply>, or <notification> elements. The namespace for <rpc> and <rpc-reply> is "ietf-netconf", but no module name at all exists for the <notification> element.

A.3. config-id attribute

Should the "config-id" (etag for the running datastore root) be returned in every <get2> response or only if requested? (Currently only if requested.)

A.4. <get2> nodeset retrieval

Should there be a retrieval mode for <get2> where only the nodes in an XPath node-set are returned? NETCONF returns all ancestor nodes and all ancestor or sibling key leafs as well. Sometime the XPath designer knows the context of the result node-set (e.g. path expression for 1 instance of a nested list). The XML scaffolding can add a lot of extra bytes to the <rpc-reply>.

Appendix B. Additional Examples

B.1. YANG Module Used in Examples

The "example-ex" YANG module models a collection of forests. Each forest has a collection of trees. For simplicity, only 1 tree of each type is allowed in a forest.

```
+--rw forests
   +--rw forest [name]
      +--rw name          string
      +--ro tree-count?   uint32
      +--rw trees
         +--rw tree [name]
            +--rw name        string
            +--rw location?   string
            +--ro height?     decimal64
```

```
module example-ex {

  namespace "http://example.com/ns/example-ex";
  prefix ex;
  organization "Example, Inc.";
  contact "support@example.com";

  description "Module used in NETCONF-EX examples.";
  revision 2013-10-19 {
    description "Initial version";
    reference "Example Spec 12.44";
  }

  container forests {
    description "A collection of forests";

    list forest {
      key name;
      description "A single forest";

      leaf name {
        type string;
        description "The forest name";
      }

      leaf tree-count {
        type uint32;
        config false;
        description "The number of trees in this forest";
```

```
         }

         container trees {
           description "A collection of trees";

           list tree {
             key name;
             description "A single tree";

             leaf name {
               type string;
               description "The tree name";
             }
             leaf location {
               type string;
               description "The tree location";
             }

             leaf height {
               type decimal64 {
                 fraction-digits 3;
               }
               units meters;
               config false;
               description "The tree height";
             }
           }  // list tree
         }  // container trees
       } // list forest
     }  // container forests
   }
```

B.2. YANG Data Used in Examples

The follow instances are assumed in the following examples.

```
     list forest: "north":
       list tree: "birch", "ash", "maple"

     list forest: "south":
       list tree: "banyan", "palm"

     leaf "location": "hillside", "west valley", "southwest pasture",
                      "east meadow", "greenhouse"
```

The forests and trees are configured, which represent trees the company has planted and growing over time.

The operational data (tree height) represents the data that the company monitors for each tree over time.

B.3. <edit2> Examples

B.3.1. Confirmed Commit on the "running" Datastore

In this example, the server supports the :writable-running and :startup capabilities:

```
<rpc message-id="105"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ex="http://example.com/ns/example-ex">
    <target><running/></target>
    <target-resource>
      /ex:forests/ex:forest[ex:name='north']
    </target-resource>
    <yang-patch>
      <patch-id>oak-tree-patch</patch-id>
      <comment>Create an oak tree</comment>
      <edit>
        <edit-id>oak</edit-id>
        <operation>create</operation>
        <target>/ex:trees</target>
        <value>
          <ex:tree>
            <ex:name>oak</ex:name>
            <ex:location>hillside</ex:location>
          </ex:tree>
        </value>
      </edit>
    </yang-patch>
    <with-locking/>
    <max-lock-wait>10</max-lock-wait>
    <confirmed/>
    <confirm-timeout>60</confirm-timeout>
    <persist>24ef8829a4</persist>
  </edit2>
</rpc>
```

The edit succeeds, and the "yang-patch-status" container is returned to the client with the <location> path expression of the new oak tree resource. The candidate and running datastores remain locked after this operation because a confirmed commit procedure is in progress. The startup datastore was not locked during this operation because the "nvstore-now" parameter was not provided.

```
<rpc-reply message-id="105"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <yang-patch-status
    xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ex="http://example.com/ns/example-ex">
    <patch-id>oak-tree-patch</patch-id>
    <ok/>
    <edit-status>
      <edit>
        <edit-id>oak</edit-id>
        <ok/>
      </edit>
    </edit-status>
  </yang-patch-status>
</rpc-reply>
```

After configuration verification (e.g., 20 seconds), the client decides to keep these configuration changes and sends a <complete-commit> request.

```
<rpc message-id="106"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <complete-commit
    xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <persist>24ef8829a4</persist>
  </complete-commit>
</rpc>
```

The server completes the confirmed commit procedure and returns an "ok" element to indicate success:

```
<rpc-reply message-id="106"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
</rpc-reply>
```

After the operation succeeds, the server releases all locks that were being held to allow exclusive write access for the entire confirmed commit procedure.

The client can now save the activated configuration changes to the startup configuration using the <copy-config> protocol operation, as described in RFC 6241, section 8.7.5.1.

B.3.2. Conditional Editing with "if-match" Parameter

In this example the client is going to change the location of the "palm" tree is the "south" forest. The entity tag for the tree resource is retrieved with the resource:

```
<rpc message-id="107"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ncex="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <xpath-filter> <!-- wrapped for display -->
      /ex:forests/ex:forest[ex:name='south']/ex:trees/
      ex:tree[ex:name='palm']
    </xpath-filter>
    <depth>1</depth>
    <with-metadata>ncex:etags</with-metadata>
  </get2>
</rpc>
```

The server returns a subtree containing data nodes representing the "palm" tree. The "etag" attribute is returned for this resource and its ancestors. Only the "tree" node itself, as requested with the "depth parameter.

```
<rpc-reply message-id="107"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:lm="urn:ietf:params:xml:ns:netconf:netconf-ex:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
  lm:last-modified="2012-09-09T02:00:00Z">
    <forests xmlns="http://example.com/ns/example-ex"
      lm:etag="34ef6892">
      <forest lm:etag="ef11eb99">
        <name>south</name>
        <trees lm:etag="ef11eb99">
          <tree lm:etag="3477cc82" />
        </trees>
      </forest>
    </forests>
  </data>
</rpc-reply>
```

The client then edits the list entry (e.g, reassigns tree location) but submits an "if-match" parameter with the "etag" value it received for the tree resource being edited:

```
<rpc message-id="108"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ex="http://example.com/ns/example-ex">
    <target><candidate/></target>
    <target-resource> <!-- wrapped for display -->
      /ex:forests/ex:forest[ex:name='south']/ex:trees
      /ex:tree[ex:name='palm']
    </target-resource>
    <yang-patch>
      <patch-id>move-palm-tree</patch-id>
      <comment>Move the palm tree</comment>
      <edit>
        <edit-id>palm</edit-id>
        <operation>merge</operation>
        <target>/</target>
        <value>
          <ex:location>greenhouse</ex:location>
        </value>
      </edit>
    </yang-patch>
    <if-match>3477cc82</if-match>
    <with-locking/>
    <max-lock-wait>10</max-lock-wait>
    <activate-now/>
    <nvstore-now/>
  </edit2>
</rpc>
```

In this example the tree resource has been edited by another client since the <get2> reply for this client, so the edit request is not even attempted. Instead an "operation-failed" is returned:

```
<rpc-reply message-id="108"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <yang-patch-status
    xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ex="http://example.com/ns/example-ex">
    <patch-id>move-palm-tree</patch-id>
    <errors>
      <error>
        <error-type>protocol</error-type>
        <error-tag>operation-failed</error-tag>
        <error-app-tag>precondition-failed</error-app-tag>
        <error-path>  <!-- wrapped for display -->
          /ex:forests/ex:forest[ex:name='south']/ex:trees/
          ex:tree[ex:name='palm']
        </error-path>
        <error-message xml:lang="en">
          if-match precondition failed
        </error-message>
      </error>
    </errors>
  </yang-patch-status>
</rpc-reply>
```

B.3.3. Bulk Editing with "target-resource" Parameter

In this example, the server supports the :candidate and :startup capabilities, so all 3 datastores (including running) are locked for the <edit2> operation. There is a new pine tree for each forest that is being created and sent to the greenhouse.

```
<rpc message-id="109"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ex="http://example.com/ns/example-ex">
    <target><candidate/></target>
    <target-resource>
      /ex:forests/ex:forest
    </target-resource>
    <yang-patch>
      <patch-id>pine-tree-patch</patch-id>
      <comment>Add 2 new pine trees to greenhouse</comment>
      <edit>
        <edit-id>pine</edit-id>
        <operation>create</operation>
        <target>/ex:trees</target>
        <value>
          <ex:tree>
            <ex:name>pine</ex:name>
            <ex:location>greenhouse</ex:location>
          </ex:tree>
        </value>
      </edit>
    </yang-patch>
    <with-locking/>
    <max-lock-wait>10</max-lock-wait>
    <activate-now/>
    <nvstore-now/>
  </edit2>
</rpc>
```

The edit succeeds, and the "yang-patch-status" container is returned to the client with the status information.

```
<rpc-reply message-id="109"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <yang-patch-status
    xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ex="http://example.com/ns/example-ex">
    <patch-id>pine-tree-patch</patch-id>
    <ok/>
    <edit-status>
      <edit>
        <edit-id>pine</edit-id>
        <ok/>
      </edit>
    </edit-status>
  </yang-patch-status>
</rpc-reply>
```

B.3.4. Edit Validation with "test-only" Parameter

In this example, the client is checking if it can change the location field in the "palm" tree list entry by using the "test-only" parameter:

```
<rpc message-id="110"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ex="http://example.com/ns/example-ex">
    <target><candidate/></target>
    <target-resource>
      /ex:forests/ex:forest[ex:name='south']/ex:trees
    </target-resource>
    <yang-patch>
      <patch-id>palm-tree-move</patch-id>
      <comment>Move the palm tree to riverside</comment>
      <edit>
        <edit-id>palm</edit-id>
        <operation>merge</operation>
        <target>/ex:tree/palm</target>
        <value>
          <ex:location>riverside</ex:location>
        </value>
      </edit>
    </yang-patch>
    <test-only/>
  </edit2>
</rpc>
```

Since "riverside" is not a supported location, an "invalid-value" error is returned for the requested edit operation:

```
<rpc-reply message-id="110"
   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <yang-patch-status
    xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ex="http://example.com/ns/example-ex">
    <patch-id>palm-tree-move</patch-id>
    <edit-status>
      <edit>
        <edit-id>palm</edit-id>
        <errors>
          <error>
            <error-type>protocol</error-type>
            <error-tag>invalid-value</error-tag>
            <error-path>  <!-- wrapped for display -->
              /ex:forests/ex:forest[ex:name='south']/ex:trees/
              ex:tree[ex:name='palm']
            </error-path>
            <error-message xml:lang="en">
              value is invalid
            </error-message>
          </error>
        </errors>
      </edit>
    </edit-status>
  </yang-patch-status>
</rpc-reply>
```

B.4. <get2> Examples

B.4.1. If-Modified-Since Non-Empty Filter Retrieval

In this example, the running datastore was last modified at "2012-09-09T01:43:27Z" because the forest named "north" was modified at this time.

- o The forest named "north" was last modified after the specified "if-modified-since" timestamp.

- o The forest named "south" was last modified before the specified "if-modified-since" timestamp.

- o The server maintains a last-modified timestamp for the running datastore and the "forest" list entries.

- o The client is requesting only the changed entries after 2012-09-09T01:43:27Z, so the "full-delta" parameter is set.

o The client is also requesting that timestamps be returned within the data nodes. If any part of the "forest" subtree is modified then this timestamp will be updated.

```
<rpc message-id="111"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ncex="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <subtree-filter>
      <forests xmlns="http://example.com/ns/example-ex" />
    </subtree-filter>
    <if-modified-since>2012-09-09T01:43:27Z</if-modified-since>
    <full-delta/>
    <with-metadata>ncex:timestamps</with-metadata>
  </get2>
</rpc>

<rpc-reply message-id="111"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:lm="urn:ietf:params:xml:ns:netconf:netconf-ex:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
  lm:last-modified="2012-09-09T02:00:00Z">
    <forests xmlns="http://example.com/ns/example-ex">
      <forest lm:last-modified="2012-09-09T02:00:00Z">
        <name>north</name>
        <trees>
          <tree>
            <name>birch</name>
            <location>hillside</location>
          </tree>
          <tree>
            <name>ash</name>
            <location>southwest pasture</location>
          </tree>
          <tree>
            <name>maple</name>
            <location>east meadow</location>
          </tree>
        </trees>
      </forest>
    </forests>
  </data>
</rpc-reply>
```

B.4.2. If-Modified-Since Empty Filter Retrieval

In this example the client has changed the "if-modified-since" timestamp to a time in the future.

- o No "forest" list entry has been modified since this time so an empty data node is returned.

- o Note that the "last-modified" timestamp is returned for the node representing the datastore, even though no data nodes have been modified since the specified time. This allows the client to easily retrieve the last-modified timestamp for the entire datastore.

```
<rpc message-id="112"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    xmlns:ncex="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <subtree-filter>
      <forests xmlns="http://example.com/ns/example-ex" />
    </subtree-filter>
    <if-modified-since>2012-09-09T03:43:27Z</if-modified-since>
    <with-metadata>ncex:timestamps</with-metadata>
  </get2>
</rpc>

<rpc-reply message-id="112"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:lm="urn:ietf:params:xml:ns:netconf:netconf-ex:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex"
    lm:last-modified="2012-09-09T02:00:00Z" />
</rpc-reply>
```

B.4.3. Keys Only Filter Retrieval

This example retrieves only the names from the "forests" subtree in the running datastore.

- o The default source (running) is used.

- o The default depth="0" is used to retrieve all subtree levels.

- o The "keys-only" leaf is set

- o The "forests" subtree is selected. The xpath-filter is used instead of the subtree-filter.

o Whitespace added to xpath-filter element for display purposes only.

```
<rpc message-id="113"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <xpath-filter xmlns:ex=http://example.com/ns/example-ex">
      /ex:forests
    </xpath-filter>
    <keys-only />
  </get2>
</rpc>

<rpc-reply message-id="113"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <forests xmlns="http://example.com/ns/example-ex">
      <forest>
        <name>north</name>
        <trees>
          <tree>
            <name>birch</name>
          </tree>
          <tree>
            <name>ash</name>
          </tree>
          <tree>
            <name>maple</name>
          </tree>
        </trees>
      </forest>
      <forest>
        <name>south</name>
        <trees>
          <tree>
            <name>banyan</name>
          </tree>
          <tree>
            <name>palm</name>
          </tree>
        </trees>
      </forest>
    </forests>
  </data>
</rpc-reply>
```

B.4.4. Test for Node Existence with Depth=1

This example retrieves the "trees" node to determine which forests have any trees.

- Only 1 subtree level is requested, instead of the default of all levels.
- The default source (running) is used.
- The "trees" subtree is selected.
- The depth parameter is set to "1" to only retrieve the requested layer "trees" and its ancestor nodes and the configuration leaf nodes from each "forest" entry.

```
<rpc message-id="114"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <subtree-filter>
      <forests xmlns="http://example.com/ns/example-ex">
        <forest>
          <trees />
        </forest>
      </forests>
    </subtree-filter>
    <depth>1</depth>
  </get2>
</rpc>

<rpc-reply message-id="114"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <forests xmlns="http://example.com/ns/example-ex">
      <forest>
        <name>north</name>
        <trees />
      </forest>
      <forest>
        <name>south</name>
        <trees />
      </forest>
    </forests>
  </data>
</rpc-reply>
```

B.4.5. Retrieve Only Non-Configuration Data Nodes

This example retrieves only the name leafs from the "forest" list within the "forests" subtree, in the running datastore.

- o The "source" leaf is set to the "operational" data source
- o The "forests" subtree is selected

```
<rpc message-id="115"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <source><operational/></source>
    <subtree-filter>
      <forests xmlns="http://example.com/ns/example-ex" />
    </subtree-filter>
  </get2>
</rpc>

<rpc-reply message-id="115"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-ex">
    <forests xmlns="http://example.com/ns/example-ex">
      <forest>
        <name>north</name>
        <trees>
          <tree>
            <name>birch</name>
            <height>41.013</height>
          </tree>
          <tree>
            <name>ash</name>
            <height>16.523</height>
          </tree>
          <tree>
            <name>maple</name>
            <height>51.204</height>
          </tree>
        </trees>
      </forest>
      <forest>
        <name>south</name>
        <trees>
          <tree>
            <name>banyan</name>
            <height>91.433</height>
          </tree>
          <tree>
            <name>palm</name>
            <height>83.439</height>
          </tree>
        </trees>
      </forest>
    </forests>
  </data>
</rpc-reply>
```

Author's Address

Andy Bierman
YumaWorks, Inc.

Email: andy@yumaworks.com

Network Working Group A. Bierman
Internet-Draft YumaWorks
Intended status: Standards Track M. Bjorklund
Expires: August 17, 2014 Tail-f Systems
K. Watsen
Juniper Networks
R. Fernando
Cisco
February 13, 2014

RESTCONF Protocol
draft-bierman-netconf-restconf-04

Abstract

This document describes a REST-like protocol that provides a programmatic interface over HTTP for accessing data defined in YANG, using the datastores defined in NETCONF.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

There is a need for standard mechanisms to allow WEB applications to access the configuration data, operational data, data-model specific protocol operations, and notification events within a networking device, in a modular and extensible manner.

This document describes a REST-like protocol called RESTCONF, running over HTTP [RFC2616], for accessing data defined in YANG [RFC6020], using datastores defined in NETCONF [RFC6241].

The NETCONF protocol defines configuration datastores and a set of Create, Retrieve, Update, Delete (CRUD) operations that can be used to access these datastores. The YANG language defines the syntax and semantics of datastore content, operational data, protocol operations, and notification events. REST-like operations are used to access the hierarchical data within a datastore.

A REST-like API can be created that provides CRUD operations on a NETCONF datastore containing YANG-defined data. This can be done in a simplified manner, compatible with HTTP and REST-like design principles. Since NETCONF protocol operations are not relevant, the user should not need any prior knowledge of NETCONF in order to use the REST-like API.

Configuration data and state data are exposed as resources that can be retrieved with the GET method. Resources representing configuration data can be modified with the DELETE, PATCH, POST, and PUT methods. Data is encoded with either XML [W3C.REC-xml-20081126] or JSON [JSON].

Data-model specific protocol operations defined with the YANG "rpc" statement can be invoked with the POST method. Data-model specific notification events defined with the YANG "notification" statement can be accessed.

1.1. Simple Subset of NETCONF Functionality

The framework and meta-model used for a REST-like API does not need to mirror those used by the NETCONF protocol, but it needs to be compatible with NETCONF. A simplified framework and protocol is needed that utilizes the three NETCONF datastores (candidate, running, startup), but hides the complexity of multiple datastores from the client.

A simplified transaction model is needed that allows basic CRUD operations on a hierarchy of conceptual resources. This represents a limited subset of the transaction capabilities of the NETCONF protocol.

Applications that require more complex transaction capabilities might consider NETCONF instead of RESTCONF. The following transaction features are not directly provided in RESTCONF:

- o datastore locking (full or partial)
- o candidate datastore
- o startup datastore
- o validate operation
- o confirmed-commit procedure

The REST-like API is not intended to replace NETCONF, but rather provide an additional simplified interface that follows REST-like principles and is compatible with a resource-oriented device abstraction.

The following figure shows the system components:

```
+-----------+           +-----------------+
|  WEB app  | <-------> |                 |
+-----------+   HTTP    | network device  |
                        |                 |
+-----------+           |   +-----------+ |
|  NMS app  | <-------> |   | datastore | |
+-----------+  NETCONF  |   +-----------+ |
                        +-----------------+
```

1.2. Data Model Driven API

RESTCONF combines the simplicity of a REST-like API over HTTP with the predictability and automation potential of a schema-driven API.

A REST-like client using HATEOAS principles would not use any data modeling language to define the application-specific content of the API. The client would discover each new child resource as it traverses the URIs returned as Location IDs to discover the server capabilities.

This approach has 3 significant weaknesses with regards to control of complex networking devices:

- inefficient performance: configuration APIs will be quite complex and may require thousands of protocol messages to discover all the schema information. Typically the data type information has to be passed in the protocol messages, which is also wasteful overhead.

- no data model richness: without a data model, the schema-level semantics and validation constraints are not available to the application.

- no tool automation: API automation tools need some sort of content schema to function. Such tools can automate various programming and documentation tasks related to specific data models.

Data model modules such as YANG modules serve as an "API contract" that will be honored by the server. An application designer can code to the data model, knowing in advance important details about the exact protocol operations and datastore content a conforming server implementation will support.

RESTCONF provides the YANG module capability information supported by the server, in case the client wants to use it. The URIs for custom protocol operations and datastore content are predictable, based on the YANG module definitions.

Operational experience with CLI and SNMP indicates that operators learn the 'location' of specific service or device related data and do not expect such information to be arbitrary and discovered each time the client opens a management session to a server.

The RESTCONF protocol operates on a conceptual datastore defined with the YANG data modeling language. The server lists each YANG module it supports under "/restconf/modules" in the top-level API resource type, using a structure based on the YANG module capability URI format defined in [RFC6020].

The conceptual datastore contents, data-model-specific operations and notification events are identified by this set of YANG module resources. All RESTCONF content identified as either a data resource, operation resource, or event stream resource is defined with the YANG language.

The classification of data as configuration or non-configuration is derived from the YANG "config" statement. Data ordering behavior is derived from the YANG "ordered-by" statement.

The RESTCONF datastore editing model is simple and direct, similar to the behavior of the ":writable-running" capability in NETCONF. Each RESTCONF edit of a datastore resource is activated upon successful completion of the transaction.

1.3. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

1.3.1. NETCONF

The following terms are defined in [RFC6241]:

- candidate configuration datastore
- client
- configuration data
- datastore
- configuration datastore
- protocol operation
- running configuration datastore
- server
- startup configuration datastore
- state data
- user

1.3.2. HTTP

The following terms are defined in [RFC2616]:

- entity tag
- fragment
- header line
- message body

- o method
- o path
- o query
- o request URI
- o response body

1.3.3. YANG

The following terms are defined in [RFC6020]:

- o container
- o data node
- o key leaf
- o leaf
- o leaf-list
- o list
- o presence container (or P-container)
- o RPC operation (now called protocol operation)
- o non-presence container (or NP-container)
- o ordered-by system
- o ordered-by user

1.3.4. Terms

The following terms are used within this document:

- o API resource: a resource with the media type "application/yang.api+xml" or "application/yang.api+json". API resources can only be edited by the server.

- o data resource: a resource with the media type "application/yang.data+xml" or "application/yang.data+json". Data resources can be edited by clients or the server. All YANG data node types can be data resources. YANG terminal nodes cannot contain sub-resources.

- datastore resource: a resource with the media type "application/yang.datastore+xml" or "application/yang.datastore+json". Represents a configuration datastore.

- edit operation: a RESTCONF operation on a data resource using the POST, PUT, PATCH, or DELETE method.

- event stream resource: a resource with the media type "application/yang.stream+xml" or "application/yang.stream+json". This resource represents an SSE (Server-Sent Events) event stream. The content consists of text using the media type "text/event-stream", as defined by the HTML5 specification. Each event represents one <notification> message generated by the server. It contains a conceptual system or data-model specific event that is delivered within a notification event stream.

- operation: the conceptual RESTCONF operation for a message, derived from the HTTP method, request URI, headers, and message body.

- operation resource: a resource with the media type "application/yang.operation+xml" or "application/yang.operation+json".

- patch: a generic PATCH method on the target datastore or data resource. The media type of the message body content will identify the patch type in use.

- plain patch: a PATCH method where the media type is "application/yang.data+xml" or "application/yang.data+json".

- query parameter: a parameter (and its value if any), encoded within the query component of the request URI.

- requested data nodes: the set of data resources identified by the target resource, or the "select" query parameter if it is present.

- resource: a conceptual object representing a manageable component within a device. Refers to the resource itself of the resource and all its sub-resources.

- retrieval request: a request using the GET or HEAD methods.

- target resource: the resource that is associated with a particular message, identified by the "path" component of the request URI.

- unified datastore: A conceptual representation of the device running configuration. The server will hide all NETCONF datastore details for edit operations, such as the ":candidate" and ":startup" capabilities.

- YANG schema resource: a resource with the media type "application/yang". The YANG representation of the schema can be retrieved by the client with the GET method.

- YANG terminal node: a YANG node representing a leaf, leaf-list, or anyxml definition.

1.3.5. Tree Diagrams

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

- Brackets "[" and "]" enclose list keys.

- 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 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. Operations

The RESTCONF protocol uses HTTP methods to identify the CRUD operation requested for a particular resource. The following table shows how the RESTCONF operations relate to NETCONF protocol operations:

RESTCONF	NETCONF
OPTIONS	none
HEAD	none
GET	<get-config>, <get>
POST	<edit-config> (operation="create")
PUT	<edit-config> (operation="replace")
PATCH	<edit-config> (operation="merge")
DELETE	<edit-config> (operation="delete")

Table 1: CRUD Methods in RESTCONF

The NETCONF "remove" operation attribute is not supported by the HTTP DELETE method. The resource must exist or the DELETE method will fail. The PATCH method is equivalent to a "merge" operation for a plain PATCH method.

Access control mechanims may be used to limit what operations can be used. In particular, RESTCONF is compatible with the NETCONF Access Control Model (NACM) [RFC6536], as there is a specific mapping between RESTCONF and NETCONF operations, defined in Table 1. The resource path needs to be converted internally by the server to the corresponding

YANG instance-identifier. Using this information, the server can apply the NACM access control rules to RESTCONF messages.

The server MUST NOT allow any operation to any resources that the client is not authorized to access.

Implementation of all methods (except PATCH) are defined in [RFC2616]. This section defines the RESTCONF protocol usage for each HTTP method.

2.1. OPTIONS

The OPTIONS method is sent by the client to discover which methods are supported by the server for a specific resource. If supported, it SHOULD be implemented for all media types. The server SHOULD implement this method, however the same information could be extracted from the YANG modules and the RESTCONF protocol specification.

2.2. HEAD

The HEAD method is sent by the client to retrieve just the headers that would be returned for the comparable GET method, without the response body. It is supported for all resource types, except operation resources.

The request MUST contain a request URI that contains at least the entry point component. The same query parameters supported by the GET method are supported by the HEAD method.

The access control behavior is enforced as if the method was GET instead of HEAD. The server MUST respond the same as if the method was GET instead of HEAD, except that no response body is included.

2.3. GET

The GET method is sent by the client to retrieve data and meta-data for a resource. It is supported for all resource types, except operation resources. The request MUST contain a request URI that contains at least the entry point component.

The server MUST NOT return any data resources for which the user does not have read privileges. If the user is not authorized to read any portion of the target resource, an error response containing a "403 Forbidden" Status-Line is returned to the client.

If the user is authorized to read some but not all of the target resource, the unauthorized content is omitted from the response message body, and the authorized content is returned to the client.

Example:

The client might request the response headers for a JSON representation of the "library" resource:

```
GET /restconf/data/example-jukebox:jukebox/
  library/artist/Foo%20Fighters/album  HTTP/1.1
Host: example.com
Accept: application/yang.data+json
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:02:40 GMT
Server: example-server
Content-Type: application/yang.data+json
Cache-Control: no-cache
Pragma: no-cache
ETag: a74eefc993a2b
Last-Modified: Mon, 23 Apr 2012 11:02:14 GMT

{
  "album" : {
    "name" : "Wasting Light",
    "genre" : "example-jukebox:alternative",
    "year" : 2011
  }
}
```

Refer to @ex-create@ for more resource creation examples.

2.4. POST

The POST method is sent by the client to create a data resource or invoke an operation resource. The server uses the target resource media type to determine how to process the request.

Type	Description
Datastore	Create a top-level configuration data resource
Data	Create a configuration data sub-resource
Operation	Invoke a protocol operation

Resource Types that Support POST

The request MUST contain a request URI that contains a target resource which identifies a datastore, data, or operation resource type.

2.4.1. Create Resource Mode

If the target resource type is a datastore or data resource, then the POST is treated as a request to create a resource or sub-resource. The message body is expected to contain the content of a child resource to create within the parent (target resource).

The "insert" and "point" query parameters are supported by the POST method for datastore and data resource types, as specified in the

YANG definition in Section 7.

If the POST method succeeds, a "201 Created" Status-Line is returned and there is no response message body. A "Location" header identifying the child resource that was created MUST be present in the response in this case.

If the user is not authorized to create the target resource, an error response containing a "403 Forbidden" Status-Line is returned to the client. All other error responses are handled according to the procedures defined in Section 6.

Example:

To create a new "jukebox" resource, the client might send:

```
POST /restconf/data HTTP/1.1
Host: example.com
Content-Type: application/yang.data+json

{ "example-jukebox:jukebox" : [null] }
```

If the resource is created, the server might respond as follows:

```
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Location: http://example.com/restconf/data/example-jukebox:jukebox
Last-Modified: Mon, 23 Apr 2012 17:01:00 GMT
ETag: b3a3e673be2
```

2.4.2. Invoke Operation Mode

If the target resource type is an operation resource, then the POST method is treated as a request to invoke that operation. The message body (if any) is processed as the operation input parameters. Refer to Section 4.6 for details on operation resources.

If the POST method succeeds, a "200 OK" Status-Line is returned if there is a response message body, and a "204 No Content" Status-Line is returned if there is no response message body.

If the user is not authorized to invoke the target operation, an error response containing a "403 Forbidden" Status-Line is returned to the client. All other error responses are handled according to the procedures defined in Section 6.

Example:

In this example, the client is invoking the "play" operation defined in the "example-jukebox" YANG module.

A client might send a "play" request as follows:

```
POST /restconf/operations/example-jukebox:play   HTTP/1.1
Host: example.com
Content-Type: application/yang.operation+json

{
  "example-jukebox:input" : {
    "playlist" : "Foo-One",
    "song-number" : 2
  }
}
```

The server might respond:

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:50:00 GMT
Server: example-server
```

2.5. PUT

The PUT method is sent by the client to create or replace the target resource.

The request MUST contain a request URI that contains a target resource that identifies the data resource to create or replace.

If the resource instance does not exist, and it represents a valid instance the server could create with a POST request, then the server SHOULD create it.

The message body is expected to contain the content used to create or replace the target resource.

The "insert" and "point" query parameters are supported by the PUT method for data resources, as specified in the YANG definition in Section 7.

Consistent with [RFC2616], if the PUT method creates a new resource, a "201 Created" Status-Line is returned. If an existing resource is modified, either "200 OK" or "204 No Content" are returned.

If the user is not authorized to create or replace the target resource an error response containing a "403 Forbidden" Status-Line is returned to the client. All other error responses are handled according to the procedures defined in Section 6.

Example:

An "album" sub-resource defined in the "example-jukebox" YANG module is replaced or created if it does not already exist.

To replace the "album" resource contents, the client might send as follows. Note that the request URI header line is wrapped for display purposes only:

```
PUT /restconf/data/example-jukebox:jukebox/
   library/artist/Foo%20Fighters/album/Wasting%20Light   HTTP/1.1
Host: example.com
Content-Type: application/yang.data+json

{
  "example-jukebox:album" : {
    "name" : "Wasting Light",
    "genre" : "example-jukebox:alternative",
    "year" : 2011
  }
}
```

If the resource is updated, the server might respond:

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:04:00 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 17:04:00 GMT
ETag: b27480aeda4c
```

2.6. PATCH

The PATCH method uses the HTTP PATCH method defined in [RFC5789] to provide an extensible framework for resource patching mechanisms. It is optional to implement by the server. Each patch type needs a unique media type. Zero or more PATCH media types MAY be supported by the server.

The "plain patch" PATCH method is used to create or update a sub-resource within the target resource. If the target resource instance does not exist, the server MUST NOT create it.

If the PATCH method succeeds, a "200 OK" Status-Line is returned if there is a message body, and "204 No Content" is returned if no response message body is sent.

If the user is not authorized to alter the target resource an error response containing a "403 Forbidden" Status-Line is returned to the client. All other error responses are handled according to the procedures defined in Section 6.

Example:

To replace just the "year" field in the "album" resource (instead of replacing the entire resource with the PUT method), the client might send a plain patch as follows. Note that the request URI header line is wrapped for display purposes only:

```
PATCH /restconf/data/example-jukebox:jukebox/
   library/artist/Foo%20Fighters/album/Wasting%20Light HTTP/1.1
Host: example.com
Content-Type: application/yang.data+json

{
  "example-jukebox:album" : {
    "genre" : "example-jukebox:rock",
    "year" : 2011
  }
}
```

If the field is updated, the server might respond:

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:49:30 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 17:49:30 GMT
ETag: b2788923da4c
```

The XML encoding for the same request might be:

```
PATCH /restconf/data/example-jukebox:jukebox/
   library/artist/Foo%20Fighters/album/Wasting%20Light HTTP/1.1
Host: example.com
If-Match: b8389233a4c
Content-Type: application/yang.data+xml


<album xmlns="http://example.com/ns/example-jukebox">
   <genre>example-jukebox:rock</genre>
   <year>2011</year>
</album>
```

2.7. DELETE

The DELETE method is used to delete the target resource. If the DELETE method succeeds, a "204 No Content" Status-Line is returned, and there is no response message body.

If the user is not authorized to delete the target resource then an error response containing a "403 Forbidden" Status-Line is returned to the client. All other error responses are handled according to the procedures defined in Section 6.

Example:

To delete a resource such as the "album" resource, the client might send:

```
DELETE /restconf/data/example-jukebox:jukebox/
   library/artist/Foo%20Fighters/album/Wasting%20Light HTTP/1.1
Host: example.com
```

If the resource is deleted, the server might respond:

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:49:40 GMT
Server: example-server
```

2.8. Query Parameters

Each RESTCONF operation allows zero or more query parameters to be present in the request URI. The specific parameters that are allowed depends on the resource type, and sometimes the specific target resource used, in the request.

Name	Methods	Description
content	GET	Select config and/or non-config data resources
depth	GET	Request limited sub-tree depth in the reply content
filter	GET	Boolean notification filter for event-stream resources
insert	POST, PUT	Insertion mode for user-ordered data resources
point	POST, PUT	Insertion point for user-ordered data resources
select	GET	Request a subset of the target resource contents

start-time	GET	Replay buffer start time for event-stream resources
stop-time	GET	Replay buffer stop time for event-stream resources

RESTCONF Query Parameters

Query parameters can be given in any order. Each parameter can appear at most once in a request URI. A default value may apply if the parameter is missing.

The semantics and syntax for all query parameters are defined in the "query-parameters" YANG grouping in Section 7. The YANG encoding MUST be converted to URL-encoded string for use in the request URI.

Refer to Appendix D.3 for examples of query parameter usage.

3. Messages

The RESTCONF protocol uses HTTP entities for messages. A single HTTP message corresponds to a single protocol method. Most messages can perform a single task on a single resource, such as retrieving a resource or editing a resource. The exception is the PATCH method, which allows multiple datastore edits within a single message.

3.1. Request URI Structure

Resources are represented with URIs following the structure for generic URIs in [RFC3986].

A RESTCONF operation is derived from the HTTP method and the request URI, using the following conceptual fields:

```
     <OP> /restconf/<path>?<query>#<fragment>


      ^       ^        ^       ^         ^
      |       |        |       |         |
    method  entry  resource  query    fragment

      M       M        O       O         I

    M=mandatory, O=optional, I=ignored


    <text> replaced by client with real values
```

- o method: the HTTP method identifying the RESTCONF operation requested by the client, to act upon the target resource specified in the request URI. RESTCONF operation details are described in Section 2.

- o entry: the well-known RESTCONF entry point ("/restconf").

- o resource: the path expression identifying the resource that is being accessed by the operation. If this field is not present, then the target resource is the API itself, represented by the media type "application/yang.api".

- o query: the set of parameters associated with the RESTCONF message. These have the familiar form of "name=value" pairs. There is a specific set of parameters defined, although the server MAY choose to support additional parameters not defined in this document. The contents of the any query parameter value MUST be encoded according to [RFC2396], section 3.4. Any reserved characters MUST be encoded with escape sequences, according to [RFC2396], section 2.4.

- fragment: This field is not used by the RESTCONF protocol.

When new resources are created by the client, a "Location" header is returned, which identifies the path of the newly created resource. The client MUST use this exact path identifier to access the resource once it has been created.

The "target" of an operation is a resource. The "path" field in the request URI represents the target resource for the operation.

3.2. Message Headers

There are several HTTP header lines utilized in RESTCONF messages. Messages are not limited to the HTTP headers listed in this section.

HTTP defines which header lines are required for particular circumstances. Refer to each operation definition section in Section 2 for examples on how particular headers are used.

There are some request headers that are used within RESTCONF, usually applied to data resources. The following tables summarize the headers most relevant in RESTCONF message requests:

Name	Description
Accept	Response Content-Types that are acceptable
Content-Type	The media type of the request body
Host	The host address of the server
If-Match	Only perform the action if the entity matches ETag
If-Modified-Since	Only perform the action if modified since time
If-Unmodified-Since	Only perform the action if un-modified since time

RESTCONF Request Headers

The following tables summarize the headers most relevant in RESTCONF message responses:

Name	Description
Allow	Valid actions when 405 error returned
Cache-Control	The cache control parameters for the response
Content-Type	The media type of the response body
Date	The date and time the message was sent
ETag	An identifier for a specific version of a resource
Last-Modified	The last modified date and time of a resource
Location	The resource identifier for a newly created resource

RESTCONF Response Headers

3.3. Message Encoding

RESTCONF messages are encoded in HTTP according to RFC 2616. The "utf-8" character set is used for all messages. RESTCONF message content is sent in the HTTP message body.

Content is encoded in either JSON or XML format.

XML encoding rules for data nodes are defined in [RFC6020]. The same encoding rules are used for all XML content.

JSON encoding rules are defined in [I-D.lhotka-netmod-json]. This encoding is valid JSON, but also has special encoding rules to identify module namespaces and provide consistent type processing of YANG data.

Request input content encoding format is identified with the Content-Type header. This field MUST be present if a message body is sent by the client.

Response output content encoding format is identified with the Accept header in the request, or if is not specified, the request input encoding format is used. If there was no request input, then the default output encoding is XML. File extensions encoded in the request are not used to identify format encoding.

3.4. RESTCONF Meta-Data

The RESTCONF protocol needs to retrieve the same meta-data that is used in the NETCONF protocol. Information about default leafs, last-modified timestamps, etc. are commonly used to annotate representations of the datastore contents. This meta-data is not defined in the YANG schema because it applies to the datastore, and is common across all data nodes.

This information is encoded as attributes in XML, but JSON does not have a standard way of attaching non-schema defined meta-data to a resource.

3.4.1. JSON Encoding of RESTCONF Meta-Data

The YANG to JSON mapping [I-D.lhotka-netmod-json] does not support attributes because YANG does not support meta-data in data node definitions. This section specifies how RESTCONF meta-data is encoded in JSON.

Only simple meta-data is supported:

- A meta-data instance can appear 0 or 1 times for a particular data node

- A meta-data instance associated with a resource is encoded as if it were a YANG leaf of type "string", according to the encoding rules in [I-D.lhotka-netmod-json], except the identifier is prepended with a "@" (%40) character.

- A meta-data instance associated with a YANG leaf or leaf-list within a resource is encoded as if it were a container for the meta-data values and the resource value in its native encoding. It is encoded according to the rules in [I-D.lhotka-netmod-json], except the meta-data identifiers are prepended with a "@" (%40) character. The resource name/value pair is repeated inside this container, which contains the actual value of the resource.

Example:

Meta-data:

```
enabled=<boolean>
owner=<owner-name>
```

YANG module: example
YANG example:

```
container top {
  leaf A {
    type int32;
  }
  leaf B {
    type boolean;
  }
}
```

The client is retrieving the "top" data resource, and the server is including datastore meta-data. Note that a query parameter to request or suppress specific meta-data is not provided in RESTCONF.

```
GET /restconf/data/example:top HTTP/1.1
Host: example.com
Accept: application/yang.data+json
```

The server might respond as follows:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Content-Type: application/yang.data+json

{
  "example:top": {
    "@enabled" : "true",
    "@owner" : "fred",
    "A" : {
      "@enabled" : "true",
      "A" : 42
    },
    "B" : {
      "@enabled" : "false",
      "B" : true
    }
  }
}
```

3.5. Return Status

Each message represents some sort of resource access. An HTTP "Status-Line" header line is returned for each request. If a 4xx or 5xx range status code is returned in the Status-Line, then the error information will be returned in the response, according to the format defined in Section 6.1.

3.6. Message Caching

Since the datastore contents change at unpredictable times, responses from a RESTCONF server generally SHOULD NOT be cached.

The server SHOULD include a "Cache-Control" header in every response that specifies whether the response should be cached. A "Pragma" header specifying "no-cache" MAY also be sent in case the "Cache-Control" header is not supported.

Instead of using HTTP caching, the client SHOULD track the "ETag" and/or "Last-Modified" headers returned by the server for the datastore resource (or data resource if the server supports it). A retrieval request for a resource can include the "If-None-Match" and/or "If-Modified-Since" headers, which will cause the server to return a "304 Not Modified" Status-Line if the resource has not changed. The client MAY use the HEAD method to retrieve just the message headers, which SHOULD include the "ETag" and "Last-Modified" headers, if this meta-data is maintained for the target resource.

4. Resources

The RESTCONF protocol operates on a hierarchy of resources, starting with the top-level API resource itself. Each resource represents a manageable component within the device.

A resource can be considered a collection of conceptual data and the set of allowed methods on that data. It can contain child nodes that are nested resources. The child resource types and methods allowed on them are data-model specific.

A resource has its own media type identifier, represented by the "Content-Type" header in the HTTP response message. A resource can contain zero or more nested resources. A resource can be created and deleted independently of its parent resource, as long as the parent resource exists.

All RESTCONF resources are defined in this document except datastore contents, protocol operations, and notification events. The syntax and semantics for these resource types are defined in YANG modules.

The RESTCONF resources are accessed via a set of URIs defined in this document. The set of YANG modules supported by the server will determine the additional data model specific operations, top-level data node resources, and notification event messages supported by the server.

The resources used in the RESTCONF protocol are identified by the "path" component in the request URI. Each operation is performed on a target resource.

4.1. RESTCONF Resource Types

The RESTCONF protocol defines some application specific media types to identify each of the available resource types. The following resource types are defined in RESTCONF:

Resource	Media Type
API	application/yang.api
Datastore	application/yang.datastore
Data	application/yang.data
Operation	application/yang.operation
Schema	application/yang
Stream	application/yang.stream

RESTCONF Media Types

4.2. Resource Discovery

A client SHOULD start by retrieving the top-level API resource, using the entry point URI "/restconf".

The RESTCONF protocol does not include a resource discovery mechanism. Instead, the definitions within the YANG modules advertised by the server are used to construct a predictable operation or data resource identifier.

The "depth" query parameter can be used to control how many descendant levels should be included when retrieving sub-resources. This parameter can be used with the GET method to discover sub-resources within a particular resource.

4.3. API Resource (/restconf)

The API resource contains the state and access points for the RESTCONF features. It is the top-level resource and has the media type "application/yang.api+xml" or "application/yang.api+json". It is accessible through the well-known URI "/restconf".

YANG Tree Diagram for "application/yang.api" Resource Type:

```
+--rw restconf
   +--rw data
   +--rw modules
   |  +--rw module [name revision]
   |     +--rw name         yang:yang-identifier
   |     +--rw revision     union
   |     +--rw schema?      empty
   |     +--rw namespace    inet:uri
   |     +--rw feature*     yang:yang-identifier
   |     +--rw deviation*   yang:yang-identifier
   |     +--rw submodule [name revision]
   |        +--rw name        yang:yang-identifier
   |        +--rw revision    union
   |        +--rw schema?     empty
   +--rw operations
   +--rw streams
   |  +--rw stream [name]
   |     +--rw name                        string
   |     +--rw description?                string
   |     +--rw replay-support?             boolean
   |     +--rw replay-log-creation-time?   yang:date-and-time
   |     +--rw events?                     empty
```

```
+--ro version?      enumeration
```

The "restconf" container definition in the "ietf-restconf" module defined in Section 7 is used to specify the structure and syntax of the conceptual sub-resources within the API resource.

This resource has the following child resources:

Child Resource	Description
data	Contains all data resources
modules	YANG module information
operations	Data-model specific operations
streams	Notification event streams
version	RESTCONF API version

RESTCONF Resources

4.3.1. /restconf/data

This mandatory resource represents the combined configuration and operational data resources that can be accessed by a client. It cannot be created or deleted by the client. The datastore resource type is defined in Section 4.4.

Example:

This example request by the client would retrieve only the non-configuration data nodes that exist within the "library" resource, using the "content" query parameter.

```
GET /restconf/data/example-jukebox:jukebox/library
   ?content=nonconfig  HTTP/1.1
Host: example.com
Accept: application/yang.data+json
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:30 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.data+json

{
  "example-jukebox:library" : {
     "artist-count" : 42,
     "album-count" : 59,
     "song-count" : 374
  }
}
```

4.3.2. /restconf/modules

This mandatory resource contains the identifiers for the YANG data model modules supported by the server.

The server MUST maintain a last-modified timestamp for this resource, and return the "Last-Modified" header when this resource is retrieved with the GET or HEAD methods.

The server SHOULD maintain an entity-tag for this resource, and return the "ETag" header when this resource is retrieved with the GET or HEAD methods.

4.3.2.1. /restconf/modules/module

This mandatory resource contains one list entry for each YANG data model module supported by the server. There MUST be an instance of this resource for every YANG module that is accessible via an operation resource or a data resource.

The contents of the "module" resource are defined in the "module" YANG list statement in Section 7.

The server MAY maintain a last-modified timestamp for each instance of this resource, and return the "Last-Modified" header when this resource is retrieved with the GET or HEAD methods. If not supported then the timestamp for the parent "modules" resource MAY be used instead.

The server MAY maintain an entity-tag for each instance of this resource, and return the "ETag" header when this resource is retrieved with the GET or HEAD methods. If not supported then the timestamp for the parent "modules" resource MAY be used instead.

4.3.3. /restconf/operations

This optional resource is a container that provides access to the data-model specific protocol operations supported by the server. The server MAY omit this resource if no data-model specific operations are advertised.

Any data-model specific operations defined in the YANG modules advertised by the server MAY be available as child nodes of this resource.

Operation resources are defined in Section 4.6.

4.3.4. /restconf/streams

This optional resource is a container that provides access to the notification event streams supported by the server. The server MAY omit this resource if no notification event streams are supported. The media type for this resource is "application/yang.api".

The server will populate this container with a stream list entry for each stream type it supports. Each stream contains a leaf called "events" which represents an event stream resource. The media type for this resource is "application/yang.stream".

Stream resources are defined in Section 4.8. Notifications are defined in Section 5.

4.3.5. /restconf/version

This sub-resource can be used by the client to identify the exact version of the RESTCONF protocol implemented by the server. The same server-wide response MUST be returned each time this resource is retrieved.

The value is assigned by the server when the server is started. The server MUST return the value "1.0" for this version of the RESTCONF protocol. This resource is encoded with the rules for an "enumeration" data type, using the "version" leaf definition in Section 7.

4.4. Datastore Resource

The /restconf/data subtree represents the datastore resource type, which is a collection of configuration and operational data nodes.

A "unified datastore" interface is used to simplify resource editing for the client. The RESTCONF unified datastore is a conceptual interface to the native configuration datastores that are present on the device.

The underlying NETCONF datastores (i.e., candidate, running, startup) can be used to implement the unified datastore, but the server design is not limited to the exact datastore procedures defined in NETCONF.

The "candidate" and "startup" datastores are not visible in the RESTCONF protocol. Transaction management and configuration persistence are handled by the server and not controlled by the client.

4.4.1. Edit Collision Detection

Two "edit collision detection" mechanisms are provided in RESTCONF, for datastore and data resources.

4.4.1.1. Timestamp

The last change time is maintained and the "Last-Modified" and "Date" headers are returned in the response for a retrieval request. The "If-Unmodified-Since" header can be used in edit operation requests to cause the server to reject the request if the resource has been modified since the specified timestamp.

The server MUST maintain a last-modified timestamp for this resource, and return the "Last-Modified" header when this resource is retrieved with the GET or HEAD methods. Only changes to configuration data resources within the datastore affect this timestamp.

4.4.1.2. Entity tag

A unique opaque string is maintained and the "ETag" header is returned in the response for a retrieval request. The "If-Match" header can be used in edit operation requests to cause the server to reject the request if the resource entity tag does not match the specified value.

The server MUST maintain a resource entity tag for this resource, and return the "ETag" header when this resource is retrieved with the GET or HEAD methods. The resource entity tag MUST be changed to a new previously unused value if changes to any configuration data resources within the datastore are made.

A datastore resource can only be written directly with the PATCH method. Only the configuration data resources within the datastore resource can be edited directly with all methods.]

Each RESTCONF edit of a datastore resource is saved to non-volatile storage in an implementation-specific matter by the server. There is no guarantee that configuration changes are saved immediately, or that the saved configuration is always a mirror of the running configuration.

4.5. Data Resource

A data resource represents a YANG data node that is a descendant node of a datastore resource.

For configuration data resources, the server MAY maintain a last-modified timestamp for the resource, and return the "Last-Modified" header when it is retrieved with the GET or HEAD methods. If maintained, the resource timestamp MUST be set to the current time whenever the resource or any configuration resource within the resource is altered.

For configuration data resources, the server MAY maintain a resource entity tag for the resource, and return the "ETag" header when it is retrieved as the target resource with the GET or HEAD methods. If maintained, the resource entity tag MUST be updated whenever the resource or any configuration resource within the resource is altered.

A data resource can be retrieved with the GET method. Data resources can be accessed via the "/restconf/data" entry point. This sub-tree is used to retrieve and edit data resources.

A configuration data resource can be altered by the client with some of all of the edit operations, depending on the target resource and the specific operation. Refer to Section 2 for more details on edit operations.

The resource definition version for a data resource is identified by the revision date of the YANG module containing the YANG definition for the data resource, specified in the /restconf/modules sub-tree.

4.5.1. Encoding YANG Instance Identifiers in the Request URI

In YANG, data nodes are named with an absolute XPath expression, defined in [XPath] , starting from the document root to the target resource. In RESTCONF, URL encoded Location header expressions are used instead.

The YANG "instance-identifier" (i-i) data type is represented in RESTCONF with the path expression format defined in this section.

Name	Comments
point	Insertion point is always a full i-i
path	Request URI path is a full or partial i-i

RESTCONF instance-identifier Type Conversion

The "path" component of the request URI contains the absolute path expression that identifies the target resource.

A predictable location for a data resource is important, since applications will code to the YANG data model module, which uses static naming and defines an absolute path location for all data nodes.

A RESTCONF data resource identifier is not an XPath expression. It is encoded from left to right, starting with the top-level data node, according to the "api-path" rule in Section 4.5.1.1. The node name of each ancestor of the target resource node is encoded in order, ending with the node name for the target resource.

If a data node in the path expression is a YANG list node, then the key values for the list (if any) are encoded according to the "key-value" rule. If the list node is the target resource, then the key values MAY be omitted, according to the operation. For example, the POST method to create a new data resource for a list node does not require key values to be present in the request URI.

The key leaf values for a data resource representing a YANG list MUST be encoded as follows:

- o The value of each leaf identified in the "key" statement is encoded in order.
- o All the components in the "key" statement MUST be encoded. Partial instance identifiers are not supported.
- o Each value is encoded using the "key-value" rule in Section 4.5.1.1, according to the encoding rules for the data type of the key leaf.
- o An empty string can be a valid key value (e.g., "/top/list/key1//key3").
- o The "/" character MUST be URL-encoded (i.e., "%2F").

- o All whitespace MUST be URL-encoded.

- o A "null" value is not allowed since the "empty" data type is not allowed for key leafs.

- o The XML encoding is defined in [RFC6020].

- o The JSON encoding is defined in [I-D.lhotka-netmod-json].

- o The entire "key-value" MUST be properly URL-encoded, according to the rules defined in [RFC3986].

- o resource URI values returned in Location headers for data resources MUST identify the module name, even if there are no conflicting local names when the resource is created. This ensures the correct resource will be identified even if the server loads a new module that the old client does not know about.

Examples:

```
[ lines wrapped for display purposes only ]

/restconf/data/example-jukebox:jukebox/library/
   artist/Beatles/album

/restconf/data/example-list:newlist/17
   /nextlist%2Ffoo%2Fbar%2Facme-list-ext%3Aext-leaf

/restconf/data/example-list:somelist/the%20key

/restconf/data/example-list:somelist/the%20key/address
```

4.5.1.1. ABNF For Data Resource Identifiers

The following ABNF syntax is used to construct RESTCONF path identifiers:

```
api-path = "/"  |
           ("/" api-identifier
             0*("/" (api-identifier | key-value )))

api-identifier = [module-name ":"] identifier

module-name = identifier

key-value = string

;; An identifier MUST NOT start with
;; (('X'|'x') ('M'|'m') ('L'|'l'))
identifier  = (ALPHA / "_")
              *(ALPHA / DIGIT / "_" / "-" / ".")

string = <an unquoted string>

[FIXME: the syntax for the select string is still TBD]
api-select = api-identifier
                0*("/" (api-identifier | key-value ))
```

4.5.2. Defaults Handling

NETCONF has a rather complex model for handling default values for leafs. RESTCONF attempts to avoid this complexity by restricting the operations that can be applied to a resource.

If the target of a GET method is a data node that represents a leaf that has a default value, and the leaf has not been given a value yet, the server MUST return the default value that is in use by the server.

If the target of a GET method is a data node that represents a container or list that has any sub-resources with default values, for the sub-resources that have not been given value yet, the server MAY return the default values that are in use by the server.

4.6. Operation Resource

An operation resource represents an protocol operation defined with the YANG "rpc" statement.

All operation resources share the same module namespace as any top-level data resources, so the name of an operation resource cannot conflict with the name of a top-level data resource defined within the same module.

If 2 different YANG modules define the same "rpc" identifier, then the module name MUST be used in the request URI. For example, if "module-A" and "module-B" both defined a "reset" operation, then invoking the operation from "module-A" would be requested as follows:

```
POST /restconf/operations/module-A:reset HTTP/1.1
Server example.com
```

Any usage of an operation resource from the same module, with the same name, refers to the same "rpc" statement definition. This behavior can be used to design protocol operations that perform the same general function on different resource types.

If the "rpc" statement has an "input" section, then a message body MAY be sent by the client in the request, otherwise the request message MUST NOT include a message body. If the "rpc" statement has an "output" section, then a message body MAY be sent by the server in the response. Otherwise the server MUST NOT include a message body in the response message, and MUST send a "204 No Content" Status-Line instead.

4.6.1. Encoding Operation Input Parameters

If the "rpc" statement has an "input" section, then the "input" node is provided in the message body, corresponding to the YANG data definition statements within the "input" section.

Example:

The following YANG definition is used for the examples in this section.

```
rpc reboot {
  input {
    leaf delay {
      units seconds;
      type uint32;
      default 0;
    }
    leaf message { type string; }
    leaf language { type string; }
  }
}
```

The client might send the following POST request message:

```
POST /restconf/operations/example-ops:reboot HTTP/1.1
Host: example.com
Content-Type: application/yang.operation+json

{
  "example-ops:input" : {
    "delay" : 600,
    "message" : "Going down for system maintenance",
    "language" : "en-US"
  }
}
```

The server might respond:

```
HTTP/1.1 204 No Content
Date: Mon, 25 Apr 2012 11:01:00 GMT
Server: example-server
```

4.6.2. Encoding Operation Output Parameters

If the "rpc" statement has an "output" section, then the "output" node is provided in the message body, corresponding to the YANG data definition statements within the "output" section.

Example:

The following YANG definition is used for the examples in this section.

```
rpc get-reboot-info {
  output {
    leaf reboot-time {
      units seconds;
      type uint32;
    }
    leaf message { type string; }
    leaf language { type string; }
  }
}
```

The client might send the following POST request message:

```
POST /restconf/operations/example-ops:get-reboot-info HTTP/1.1
Host: example.com
Accept: application/yang.operation+json
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 25 Apr 2012 11:10:30 GMT
Server: example-server
Content-Type: application/yang.operation+json

{
  "example-ops:output" : {
    "reboot-time" : 30,
    "message" : "Going down for system maintenance",
    "language" : "en-US"
  }
}
```

4.7. Schema Resource

If the server supports the "schema" leaf within the API then the client can retrieve the YANG schema text for the associated YANG module or submodule, using the GET method.

The client might send the following GET request message:

```
GET /restconf/modules/module/example-jukebox/2013-12-21/schema
   HTTP/1.1
Host: example.com
Accept: application/yang
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 25 Apr 2012 11:10:30 GMT
Server: example-server
Content-Type: application/yang

module example-jukebox {

   namespace "http://example.com/ns/example-jukebox";
   prefix "jbox";

   // rest of YANG module content deleted...
}
```

4.8. Stream Resource

A stream resource represents a source for system generated event notifications. Each stream is created and modified by the server only. A client can retrieve a stream resource or initiate a long-poll server sent event stream, using the procedure specified in Section 5.3.

A notification stream functions according to the NETCONF Notifications specification [RFC5277]. The "ietf-restconf" YANG module contains the "stream" list (/restconf/streams/stream) which specifies the syntax and semantics of a stream resource.

5. Notifications

The RESTCONF protocol supports YANG-defined event notifications. The solution preserves aspects of NETCONF Event Notifications [RFC5277] while utilizing the Server-Sent Events [wd-eventsource] transport strategy.

5.1. Server Support

A RESTCONF server is not required to support RESTCONF notifications. Clients may determine if a server supports RESTCONF notifications by using the HTTP operation OPTIONS, HEAD, or GET on the "/restconf/streams" resource described below. The server does not support RESTCONF notifications if an HTTP error code is returned (e.g. 404 Not Found).

5.2. Event Streams

A RESTCONF server that supports notifications will populate a stream resource for each notification delivery service access point. A RESTCONF client can retrieve the list of supported event streams from a RESTCONF server using the GET operation on the "/restconf/streams" resource.

The "/restconf/streams" container definition in the "ietf-restconf" module defined in Section 7 is used to specify the structure and syntax of the conceptual sub-resources within the "streams" resource.

For example:

The client might send the following request:

```
GET /restconf/streams HTTP/1.1
Host: example.com
Accept: application/yang.api+xml
```

The server might send the following response:

```
HTTP/1.1 200 OK
Content-Type: application/yang.api+xml
```

```
<streams xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
   <stream>
      <name>NETCONF</name>
      <description>default NETCONF event stream
      </description>
      <replay-support>true</replay-support>
      <replay-log-creation-time>
         2007-07-08T00:00:00Z
      </replay-log-creation-time>
      <events/>
   </stream>
   <stream>
      <name>SNMP</name>
      <description>SNMP notifications</description>
      <replay-support>false</replay-support>
      <events/>
   </stream>
   <stream>
      <name>syslog-critical</name>
      <description>Critical and higher severity
      </description>
      <replay-support>true</replay-support>
      <replay-log-creation-time>
         2007-07-01T00:00:00Z
      </replay-log-creation-time>
      <events/>
   </stream>
</streams>
```


5.3. Subscribing to Receive Notifications

RESTCONF clients can subscribe to receive notifications by sending an HTTP GET request for the "/restconf/streams/stream/<stream-name>" resource, with the "Accept" type "text/event-stream". The server will treat the connection as an event stream, using the Server Sent Events [wd-eventsource] transport strategy.

The server MAY support query parameters for a GET method on this resource. These parameters are specific to each notification stream.

For example:

```
GET /restconf/streams/stream/NETCONF/events HTTP/1.1
Host: example.com
Accept: text/event-stream
Cache-Control: no-cache
Connection: keep-alive
```

A RESTCONF client MAY request the server compress the events using the HTTP header field "Accept-Encoding". For instance:

```
GET /restconf/streams/stream/NETCONF/events HTTP/1.1
Host: example.com
Accept: text/event-stream
Cache-Control: no-cache
Connection: keep-alive
Accept-Encoding: gzip, deflate
```

5.3.1. NETCONF Event Stream

The server SHOULD support the "NETCONF" notification stream defined in [RFC5277]. For this stream, RESTCONF notification subscription requests MAY specify parameters indicating the events it wishes to receive.

Name	Description
start-time	replay event start time
stop-time	replay event stop time
filter	boolean content filter

NETCONF Stream Query Parameters

The semantics and syntax for these query parameters are defined in the "query-parameters" YANG grouping in Section 7. The YANG encoding MUST be converted to URL-encoded string for use in the request URI.

Refer to Appendix D.3.3 for filter parameter examples.

5.4. Receiving Event Notifications

RESTCONF notifications are encoded according to the definition of the event stream. The NETCONF stream defined in [RFC5277] is encoded in XML format.

The structure of the event data is based on the "notification" element definition in section 4 of [RFC5277]. It MUST conform to the "notification" YANG container definition in Section 7.

An example SSE notification encoded using XML:

```
data: <notification
data:    xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
data:    <event-time>2013-12-21T00:01:00Z</event-time>
data:    <event xmlns="http://example.com/event/1.0">
data:       <eventClass>fault</eventClass>
data:       <reportingEntity>
data:           <card>Ethernet0</card>
data:       </reportingEntity>
data:       <severity>major</severity>
data:     </event>
data: </notification>
```

Since XML is not whitespace sensitive, the above message can be encoded onto a single line.

For example: ('\' line wrapping added for formatting only)

```
data: <notification xmlns="urn:ietf:params:xml:ns:yang:ietf-rest\
conf"><event-time>2013-12-21T00:01:00Z</event-time><event xmlns="\
http://example.com/event/1.0"><eventClass>fault</eventClass><repo\
rtingEntity><card>Ethernet0</card></reportingEntity><severity>maj\
or</severity></event></notification>
```

The SSE specifications supports the following additional fields: event, id and retry. A RESTCONF server MAY send the "retry" field and, if it does, RESTCONF clients SHOULD use it. A RESTCONF server SHOULD NOT send the "event" or "id" fields, as there are no meaningful values that could be used for them that would not be redundant to the contents of the notification itself. RESTCONF servers that do not send the "id" field also do not need to support the HTTP header "Last-Event-Id". RESTCONF servers that do send the "id" field MUST still support the "startTime" query parameter as the preferred means for a client to specify where to restart the event stream.

6. Error Reporting

HTTP Status-Lines are used to report success or failure for RESTCONF operations. The <rpc-error> element returned in NETCONF error responses contains some useful information. This error information is adapted for use in RESTCONF, and error information is returned for "4xx" class of status codes.

The following table summarizes the return status codes used specifically by RESTCONF operations:

Status-Line	Description
100 Continue	POST accepted, 201 should follow
200 OK	Success with response body
201 Created	POST to create a resource success
202 Accepted	POST to create a resource accepted
204 No Content	Success without response body
304 Not Modified	Conditional operation not done
400 Bad Request	Invalid request message
403 Forbidden	Access to resource denied
404 Not Found	Resource target or resource node not found
405 Method Not Allowed	Method not allowed for target resource
409 Conflict	Resource or lock in use
412 Precondition Failed	Conditional method is false
413 Request Entity Too Large	too-big error
414 Request-URI Too Large	too-big error
415 Unsupported Media Type	non RESTCONF media type
500 Internal Server Error	operation-failed
501 Not Implemented	unknown-operation
503 Service Unavailable	Recoverable server error

HTTP Status Codes used in RESTCONF

Since an operation resource is defined with a YANG "rpc" statement, a mapping between the NETCONF <error-tag> value and the HTTP status code is needed. The specific error condition and response code to use are data-model specific and might be contained in the YANG "description" statement for the "rpc" statement.

<error-tag>	status code
in-use	409
invalid-value	400
too-big	413
missing-attribute	400
bad-attribute	400
unknown-attribute	400
bad-element	400
unknown-element	400
unknown-namespace	400
access-denied	403
lock-denied	409
resource-denied	409
rollback-failed	500
data-exists	409
data-missing	409
operation-not-supported	501
operation-failed	500
partial-operation	500
malformed-message	400

Mapping from error-tag to status code

6.1. Error Response Message

When an error occurs for a request message on a data resource or an operation resource, and a "4xx" class of status codes (except for status code "403"), then the server SHOULD send a response body containing the information described by the "errors" container definition within the YANG module Section 7.

YANG Tree Diagram for <errors> Data:

```
+--ro errors
   +--ro error
      +--ro error-type       enumeration
      +--ro error-tag        string
      +--ro error-app-tag?   string
      +--ro (error-node)?
      |  +--:(error-path)
      |  |  +--ro error-path?      instance-identifier
      |  +--:(error-urlpath)
      |     +--ro error-urlpath?   data-resource-identifier
      +--ro error-message?   string
      +--ro error-info
```

The semantics and syntax for RESTCONF error messages are defined in the "errors" YANG grouping in Section 7.

Examples:

The following example shows an error returned for an "lock-denied" error on a datastore resource.

```
POST /restconf/operations/example-ops:lock-datastore HTTP/1.1
Host: example.com
```

The server might respond:

```
HTTP/1.1 409 Conflict
Date: Mon, 23 Apr 2012 17:11:00 GMT
Server: example-server
Content-Type: application/yang.api+json

{
  "ietf-restconf:errors": {
    "error": {
      "error-type": "protocol",
      "error-tag": "lock-denied",
      "error-message": "Lock failed, lock already held"
    }
  }
}
```

The following example shows an error returned for a "data-exists" error on a data resource. The "jukebox" resource already exists so it cannot be created.

The client might send:

```
POST /restconf/data/example-jukebox:jukebox HTTP/1.1
Host: example.com
```

The server might respond:

```
HTTP/1.1 409 Conflict
Date: Mon, 23 Apr 2012 17:11:00 GMT
Server: example-server
Content-Type: application/yang.api+json
```


```
{
  "ietf-restconf:errors": {
    "error": {
      "error-type": "protocol",
      "error-tag": "data-exists",
      "error-urlpath": "http://example.com/restconf/data/
           example-jukebox:jukebox",
      "error-message":
        "Data already exists, cannot create new resource"
    }
  }
}
```

7. RESTCONF module

The "ietf-restconf" module defines conceptual definitions within groupings, which are not meant to be implemented as datastore contents by a server.

The "ietf-yang-types" and "ietf-inet_types" modules from [RFC6991] are used by this module for some type definitions.

RFC Ed.: update the date below with the date of RFC publication and remove this note.

<CODE BEGINS> file "ietf-restconf@2014-02-13.yang"

```
module ietf-restconf {
  namespace "urn:ietf:params:xml:ns:yang:ietf-restconf";
  prefix "rc";

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

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

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

     WG Chair: Bert Wijnen
               <mailto:bertietf@bwijnen.net>

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

     Editor:   Andy Bierman
               <mailto:andy@yumaworks.com>

     Editor:   Martin Bjorklund
               <mailto:mbj@tail-f.com>

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

     Editor:   Rex Fernando
               <mailto:rex@cisco.com>";

  description
    "This module contains conceptual YANG specifications
```

```
         for the message and error content that is used in
         RESTCONF protocol messages. A conceptual container
         representing the RESTCONF API nodes is also defined
         for the media type application/yang.api.

         Note that the YANG definitions within this module do not
         represent configuration data of any kind.
         The YANG grouping statements provide a normative syntax
         for XML and JSON message encoding purposes.

         Copyright (c) 2013 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.";

      // RFC Ed.: replace XXXX with actual RFC number and remove this
      // note.

      // RFC Ed.: remove this note
      // Note: extracted from draft-bierman-netconf-restconf-04.txt

      // RFC Ed.: update the date below with the date of RFC publication
      // and remove this note.
      revision 2014-02-13 {
        description
          "Initial revision.";
        reference
          "RFC XXXX: RESTCONF Protocol.";
      }

      typedef data-resource-identifier {
        type string {
          length "1 .. max";
        }
        description
          "Contains a Data Resource Identifier formatted string
           to identify a specific data resource instance.
           The document root for all data resources is a
           datastore resource container. Each top-level YANG
           data nodes supported by the server will be represented
```

```
          as a child node of the document root.

          The canonical representation of a data resource identifier
          includes the full server specification that is returned
          in the Location header when a new data resource is created
          with the POST method.

          The abbreviated representation does not contain any server
          location identification. Instead the identifier will start
          with the '/' character to represent the datastore document
          root for the data resource instance.

          The server MUST accept either representation and SHOULD
          return the canonical representation in any response message.";
       reference
         "RFC XXXX: [sec. 5.3.1.1 ABNF For Data Resource Identifiers]";
     }

     typedef revision-identifier {
       type string {
         pattern '\d{4}-\d{2}-\d{2}';
       }
       description
         "Represents a specific date in YYYY-MM-DD format.
          TBD: make pattern more precise to exclude leading zeros.";
     }

     grouping errors {

       description
         "A grouping that contains a YANG container
          representing the syntax and semantics of a
          YANG Patch errors report within a response message.";

       container errors {
         config false;  // needed so list error does not need a key
         description
           "Represents an error report returned by the server if
            a request results in an error.";

         list error {
           description
             "An entry containing information about one
              specific error that occurred while processing
              a RESTCONF request.";
           reference "RFC 6241, Section 4.3";

           leaf error-type {
```

```
          type enumeration {
            enum transport {
              description "The transport layer";
            }
            enum rpc {
              description "The rpc or notification layer";
            }
            enum protocol {
              description "The protocol operation layer";
            }
            enum application {
              description "The server application layer";
            }
          }
          mandatory true;
          description
            "The protocol layer where the error occurred.";
        }

        leaf error-tag {
          type string;
          mandatory true;
          description
            "The enumerated error tag.";
        }

        leaf error-app-tag {
          type string;
          description
            "The application-specific error tag.";
        }

        choice error-node {
          description
            "The server will return the location of the error node
             in a format that is appropriate for the protocol.
             If no specific node within the request message body
             caused the error then this choice will not be present.";

          leaf error-path {
            type instance-identifier;
            description
              "The YANG instance identifier associated
               with the error node. This leaf will only be
               present if the error node is not a data resource,
               e.g., the error node is an input parameter
               for an operation resource.";
          }
```

```
            leaf error-urlpath {
              type data-resource-identifier;
              description
                "The target data resource identifier associated
                 with the error node.  This leaf will only be
                 present if the error node is associated with
                 a data resource (either within the server or
                 in the request message).";
            }
          }

          leaf error-message {
            type string;
            description
              "A message describing the error.";
          }

          anyxml error-info {
             description
               "Arbitrary XML that represents a container
                of additional information for the error report.";
          }
        }
      }
    } // grouping errors


    grouping restconf {
      description
        "A grouping that contains a YANG container
         representing the syntax and semantics of
         the RESTCONF API resource.";

      container restconf {
        description
          "Conceptual container representing the
           application/yang.api resource type.";

        container data {
          description
            "Container representing the application/yang.datastore
             resource type. Represents the conceptual root of all
             operational data and configuration data supported by
             the server.  The child nodes of this container can be
             any data resource (application/yang.data), which are
             defined as top-level data nodes from the YANG modules
             advertised by the server in /restconf/modules.";
        }
```

```
         container modules {
           description
             "Contains a list of module description entries.
              These modules are currently loaded into the server.";

           grouping common-leafs {
             description
               "Common parameters for YANG modules and submodules.";

             leaf name {
               type yang:yang-identifier;
               description "The YANG module or submodule name.";
             }
             leaf revision {
               type union {
                 type revision-identifier;
                 type string { length 0; }
               }
               description
                 "The YANG module or submodule revision date.
                  An empty string is used if no revision statement
                  is present in the YANG module or submodule.";
             }

             leaf schema {
               type empty;
               description
                 "Represents the YANG schema resource for this module
                  or submodule if it is available on the server.
                  This leaf will only be present if the server has
                  the schema available for retrieval.  A GET
                  request with a target resource URI that identifies
                  this leaf will cause the server to return the YANG
                  schema text for the associated module or submodule.";
             }
           }


           list module {
             key "name revision";
             description
               "Each entry represents one module currently
                supported by the server.";

             uses common-leafs;

             leaf namespace {
               type inet:uri;
```

```
              mandatory true;
              description
                "The XML namespace identifier for this module.";
            }
            leaf-list feature {
              type yang:yang-identifier;
              description
                "List of YANG feature names from this module that are
                 supported by the server.";
            }
            leaf-list deviation {
              type yang:yang-identifier;
              description
                "List of YANG deviation module names used by this
                 server to modify the conformance of the module
                 associated with this entry.";
            }

            list submodule {
              key "name revision";
              description
                "Each entry represents one submodule within the
                 parent module.";

              uses common-leafs;
            }
          }
        }

        container operations {
          description
            "Container for all operation resources
             (application/yang.operation),

             Each resource is represented as an empty leaf with the
             name of the RPC operation from the YANG rpc statement.

             E.g.;

                POST /restconf/operations/show-log-errors

                leaf show-log-errors {
                  type empty;
                }
            ";
        }

        container streams {
```

```
         description
           "Container representing the notification event streams
            supported by the server.";
          reference
            "RFC 5277, Section 3.4, <streams> element.";

         list stream {
           key name;
           description
             "Each entry describes an event stream supported by
              the server.";

           leaf name {
             type string;
             description "The stream name";
             reference "RFC 5277, Section 3.4, <name> element.";
           }

           leaf description {
             type string;
             description "Description of stream content";
             reference
               "RFC 5277, Section 3.4, <description> element.";
           }

           leaf replay-support {
             type boolean;
             description
               "Indicates if replay buffer supported for this stream";
             reference
               "RFC 5277, Section 3.4, <replaySupport> element.";
           }

           leaf replay-log-creation-time {
             type yang:date-and-time;
             description
               "Indicates the time the replay log for this stream
                was created.";
             reference
               "RFC 5277, Section 3.4, <replayLogCreationTime>
                element.";
           }

           leaf events {
             type empty;
             description
               "Represents the entry point for establishing
                notification delivery via server sent events.";
```

```
          }
        }
      }

      leaf version {
        type enumeration {
          enum "1.0" {
            description
              "Version 1.0 of the RESTCONF protocol.";
          }
        }
        config false;
        description
          "Contains the RESTCONF protocol version.";
      }
    }
  }  // grouping restconf


  grouping query-parameters {
    description
       "Contains conceptual definitions for the query string
        parameters used in the RESTCONF protocol.";

    leaf content {
      type enumeration {
        enum config {
          description
            "Return only configuration descendant data nodes";
        }
        enum nonconfig {
          description
            "Return only non-configuration descendant data nodes";
        }
        enum all {
          description
            "Return all descendant data nodes";
        }
      }
      description
         "The content parameter controls how descendant nodes of
          the requested data nodes will be processed in the reply.

          This parameter is only allowed for GET methods on
          datastore and data resources.  A 400 Bad Request error
          is returned if used for other methods or resource types.

          The default value is determined by the config-stmt
```

```
             value of the requested data nodes. If 'false', then
             the default is 'nonconfig'.  If 'true' then the
             default is 'config'.";
       }

       leaf depth {
         type union {
           type enumeration {
             enum unbounded {
               description "All sub-resources will be returned.";
             }
           }
           type uint32 {
             range "1..max";
           }
         }
         default unbounded;
         description
           "The 'depth' parameter is used to specify the number
            of nest levels returned in a response for a GET method.
            The first nest-level consists of the requested data node
            itself. Any child nodes which are contained within
            a parent node have a depth value that is 1 greater than
            its parent.

            This parameter is only allowed for GET methods on api,
            datastore, and data resources.  A 400 Bad Request error
            is returned if used for other methods or resource types.

            By default, the server will include all sub-resources
            within a retrieved resource, which have the same resource
            type as the requested resource. Only one level of
            sub-resources with a different media type than the target
            resource will be returned.";
       }

       leaf filter {
         type yang:xpath1.0;
         description
           "The 'filter' parameter is used to indicate which subset of
            all possible events are of interest.  If not present, all
            events not precluded by other parameters will be sent.

            This parameter is only allowed for GET methods on a
            text/event-stream data resource.  A 400 Bad Request error
            is returned if used for other methods or resource types.

            The format of this parameter is an XPath expression, and
```

```
            is evaluated in the following context:

            o  The set of namespace declarations is the set of
               prefix and namespace pairs for all supported YANG
               modules, where the prefix is the YANG module name, and
               the namespace is as defined by the 'namespace' statement
               in the YANG module.
            o  The function library is the core function library defined
               in XPATH.

            o  The set of variable bindings is empty.

            o  The context node is the root node

           The filter is used as defined in [RFC5277], section 3.6.
           If the boolean result of the expression is true when applied
           to the conceptual 'notification' document root, then the
           notification event is delivered to the client.";
       }

       leaf insert {
         type enumeration {
           enum first {
             description "Insert the new data as the new first entry.";
           }
           enum last {
             description "Insert the new data as the new last entry.";
           }
           enum before {
             description
               "Insert the new data before the insertion point,
                specified by the value of the 'point' parameter.";
           }
           enum after {
             description
               "Insert the new data after the insertion point,
                specified by the value of the 'point' parameter.";
           }
         }
         default last;
         description
           "The 'insert' parameter is used to specify how a
            resource should be inserted within a user-ordered list.

            This parameter is only supported for the POST and PUT
            methods. It is also only supported if the target
            resource is a data resource, and that data represents
            a YANG list or leaf-list that is ordered by the user.
```

```
             If the values 'before' or 'after' are used,
             then a 'point' query parameter for the insertion
             parameter MUST also be present, or a 400 Bad Request
             error is returned.";
        }

        leaf point {
          type data-resource-identifier;
          description
            "The 'point' parameter is used to specify the
             insertion point for a data resource that is being
             created or moved within a user ordered list or leaf-list.

             This parameter is only supported for the POST and PUT
             methods. It is also only supported if the target
             resource is a data resource, and that data represents
             a YANG list or leaf-list that is ordered by the user.

             If the 'insert' query parameter is not present, or has
             a value other than 'before' or 'after', then a 400
             Bad Request error is returned.

             This parameter contains the instance identifier of the
             resource to be used as the insertion point for a
             POST or PUT method.";
        }

        leaf select {
          type string {
            length "1 .. max";
          }
          description
            "The 'select' query parameter is used to specify an
             expression which can represent a subset of all data nodes
             within the target resource. It contains an expression
             string, using the target resource as the context node.

             The encoding for the select parameter is still TBD.

             This parameter is only allowed for GET methods on api,
             datastore, and data resources.  A 400 Bad Request error
             is returned if used for other methods or resource types.

             If XPath:
             The string is an XPath expression that will be evaluated
             using the target resource instance as the context node
             and the document root. It is expected to return a node-set
             result representing the descendants within the context
```

```
            node that should be returned in a GET response.";
       }

       leaf start-time {
         type yang:date-and-time;
         description
           "The 'start-time' parameter is used to trigger
            the notification replay feature and indicate
            that the replay should start at the time specified.
            If the stream does not support replay, per the
            'replay-support' attribute returned by
            the /restconf/streams resource, then the server MUST
            return the HTTP error code 400 Bad Request.

            This parameter is only allowed for GET methods on a
            text/event-stream data resource.  A 400 Bad Request error
            is returned if used for other methods or resource types.

            If this parameter is not present, then a replay subscription
            is not begin requested.  It is not valid to specify start
            times that are later than the current time.  If the value
            specified is earlier than the log can support, the replay
            will begin with the earliest available notification";
       }

       leaf stop-time {
         type yang:date-and-time;
         description
           "The 'stop-time' parameter is used with the
            replay feature to indicate the newest notifications of
            interest.  This parameter MUST be used with and have a
            value later than the 'start-time' parameter.

            This parameter is only allowed for GET methods on a
            text/event-stream data resource.  A 400 Bad Request error
            is returned if used for other methods or resource types.

            If this parameter is not present, the notifications will
            continue until the subscription is terminated.
            Values in the future are valid.";
       }

     }  // grouping query-parameters


     grouping notification {
       description
         "Contains the notification message wrapper definition.";
```

```
       container notification {
         description
           "RESTCONF notification message wrapper.";

         leaf event-time {
           type yang:date-and-time;
           mandatory true;
           description
             "The time the event was generated by the
              event source.";
           reference
             "RFC 5277, section 4, <eventTime> element.";
         }

         /* The YANG-specific notification container is encoded
          * after the 'event-time' element.  The format
          * corresponds to the notificationContent element
          * in RFC 5277, section 4. For example:
          *
          *  module example-one {
          *     ...
          *     notification event1 { ... }
          *
          *  }
          *
          *  Encoded as element 'event1' in the namespace
          *  for module 'example-one'.
          */
       }
     }  // grouping notification

   }

   <CODE ENDS>
```

8. IANA Considerations

8.1. YANG Module Registry

This document registers one URI in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested to be made.

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

This document registers one YANG module in the YANG Module Names registry [RFC6020].

```
name:         ietf-restconf
namespace:    urn:ietf:params:xml:ns:yang:ietf-restconf
prefix:       rc
// RFC Ed.: replace XXXX with RFC number and remove this note
reference:    RFC XXXX
```

8.2. application/yang Media Sub Types

The parent MIME media type for RESTCONF resources is application/yang, which is defined in [RFC6020]. This document defines the following sub-types for this media type.

- api
- data
- datastore
- operation
- stream

Type name: application

Subtype name: yang.xxx

Required parameters: TBD

Optional parameters: TBD

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

// RFC Ed.: replace XXXX with RFC number and remove this note

Published specification: RFC XXXX

9. Security Considerations

TBD

10. References

10.1. Normative References

[I-D.lhotka-netmod-json]
Lhotka, L., "Modeling JSON Text with YANG", draft-lhotka-netmod-yang-json-02 (work in progress), September 2013.

[JSON] Bray, T., Ed., "The JSON Data Interchange Format", draft-ietf-json-rfc4627bis-10 (work in progress), December 2013.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2396] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, July 2008.

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, March 2010.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012.

[RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013.

[W3C.REC-xml-20081126]
Yergeau, F., Maler, E., Paoli, J., Sperberg-McQueen, C., and T. Bray, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126>.

[wd-eventsource]
Hickson, I., "Server-Sent Events", December 2012.

10.2. Informative References

[XPath] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation REC-xpath-19991116, November 1999, <http://www.w3.org/TR/1999/REC-xpath-19991116>.

Appendix A. Change Log

-- RFC Ed.: remove this section before publication.

A.1. 03 to 04

- o Changed term RESTful to REST-like
- o Clarified SSE section
- o Clarified access control section
- o Clarified return code test for some HTTP methods
- o Fixed 2 examples that were wrong

A.2. 02 to 03

- o Move YANG Patch to separate document
- o Remove some introduction text
- o Move most framework text into other sections
- o Move notifications into its own section
- o Remove fields; all YANG node types are sub-resources
- o Add schema retrieval via /restconf/modules/module/schema resource
- o Add submodule support to /restconf/modules/module resource
- o Move some examples to appendix
- o Add canonical representation for data-resource-identifier typedef
- o Move query string definitions to YANG module
- o Removed "format" query parameter
- o Removed user-selected XML/JSON encoding for events. Event streams define a fixed encoding. It is not user-configurable.
- o Clarified that query parameters for SSE event streams are specific to the stream definition; start-time, stop-time, and filter only apply to the NETCONF stream;

- o Change operation input and output media type from application/yang.data to application/yang.data.

- o Fix some bugs and typos

A.3. 01 to 02

- o Added Notification Model (section 2.2)

- o Remove error-action from YANG Patch

- o Add "comment" and "ok" leafs to yang-patch-status container

- o Fixed YANG Patch JSON example syntax

- o Added stream resource type and streams container to /restconf container

- o Removed "vnd" from media type definitions

- o Changed yang-patch edit list from ascending uint32 key to an arbitrary string key and an ordered-by user list.

- o Several clarifications and corrections

- o Add YANG tree diagrams

- o Add application/yang.patch-status media type

- o Remove redundant "global-errors" container from "yang-patch-status" container

- o Split the /restconf/datastore entry point into 2 entry points (config and operational)

- o Remove the "config" parameter since it is no longer needed after datastore is split

A.4. 00 to 01

- o Removed incorrect /.well-known URI prefix.

- o Remove incorrect IANA request for well-known URI.

- o Clarified that API resource type nodes are defined in the ietf-restconf namespace.

- Changed CamelCase names in example-jukebox.yang to lowercase, and updated examples.

- Updated and corrected YANG types in ietf-restconf module.

A.5. YANG-API-01 to RESTCONF-00

- Protocol renamed from YANG-API to RESTCONF

- Fields are clarified. Containers and lists are sub-resources. All other YANG data node types are fields within a parent resource.

- The 'optional-key' YANG extension has been removed.

- The default value is returned by the server if the target resource represents a missing data node but the server is using a default value for the leaf.

- The default for the 'depth' parameter has been changed from '1' to 'unbounded'. The depth is only limited if an integer value for this parameter is specified by the client.

- The default for the 'format' parameter has been changed from 'json' to 'xml'.

- expanded introduction

- removed transactions

- removed capabilities

- removed usage of Range and IfRange headers

- simplified editing model

- removed global protocol operations from ietf-restconf.yang

- changed RPC operation terminology to protocol operation

- updated JSON draft reference

- updated IANA section

- added YANG Patch

- added YANG definitions to ietf-restconf.yang

- o added Kent Watsen and Rex Fernando as co-authors

- o updated YANG modules so they pass pyang --ietf checking

- o changed examples so resource URIs use the module name variant to identify data resources

- o changed depth behavior so the entire server contents are not returned for "GET /restconf"; Server will stop at new resource type; e.g. yang.api --> yang.datastore returns the datastore as an empty node; yang.api --> yang.operation returns the operation name as an empty node;

Appendix B. Open Issues

B.1. message-id

- There is no "message-id" field in a RESTCONF message. Is a message identifier needed? If so, should either the "Message-ID" or "Content-ID" header from RFC 2392 be used for this purpose?

B.2. select parameter

- What syntax should be used for the "select" query parameter? The current choices are "XPath" and "path-expr". Perhaps an additional parameter to identify the select string format is needed to allow extensibility?

B.3. server support verification

- Are all header lines used by RESTCONF supported by common application frameworks, such as FastCGI and WSGI? If not, then should query parameters be used instead, since the QUERY_STRING is widely available to WEB applications?

B.4. error media type

- Should the <errors> element returned in error responses be a separate media type?

B.5. additional datastores

- How should additional datastores be supported, which may be added to the NETCONF/NETMOD framework in the future?

B.6. PATCH media type discovery

- How does a client know which PATCH media types are supported by the server in addition to application/yang.data and application/yang.patch?

B.7. RESTCONF version

- Is the /restconf/version field considered meta-data? Should it be returned as XRD (Extensible Resource Descriptor)? In addition or instead of the version field? Should this be the ietf-restconf YANG module revision date, instead of the string 1.0?

B.8. YANG to resource mapping

- o Since data resources can only be YANG containers or lists, what should be done about top-level YANG data nodes that are not containers or lists? Are they allowed in RESTCONF?

- o Can a choice be a resource? YANG choices are invisible to RESTCONF at this time.

B.9. .well-known usage

- o Does RESTCONF need to Use a .well-known link relation to to re-map API entry point?

The client first discovers the server's root for the RESTCONF API. In this example, it is "/api/restconf":

```
Request
-------
GET /.well-known/host-meta users HTTP/1.1
Host: example.com
Accept: application/xrd+xml

Response
--------
HTTP/1.1 200 OK
Content-Type: application/xrd+xml
Content-Length: nnn


<XRD xmlns='http://docs.oasis-open.org/ns/xri/xrd-1.0'>
    <Link rel='restconf' href='/api/restconf'/>
</XRD>
```

Once discovering the RESTCONF API root, the client MUST prepend it to any access to a RESTCONF resource:

```
Request
-------
GET /api/restconf/version  HTTP/1.1
Host: example.com
Accept: application/yang.api+json

Response
--------
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Last-Modified: Sun, 22 Apr 2012 01:00:14 GMT
Content-Type: application/yang.api+json

{ "version": "1.0" }
```

B.10. _self links for HATEOAS support

- o Should there be a mode where the client can request that the resource identifier is returned in a GET request?

B.11. netconf-state monitoring support

- o Should long-term RESTCONF operations (i.e. SSE long-poll) be considered sessions with regards to NETCONF monitoring "session" list? If so, what text is needed in RESTCONF draft to standardize the RESTCONF session entries?

B.12. secure transport

- o Details to support secure operation over TLS are needed

- o Security considerations need to be written

Appendix C. Example YANG Module

The example YANG module used in this document represents a simple media jukebox interface.

YANG Tree Diagram for "example-jukebox" Module

```
   +--rw jukebox?
      +--rw library
      |  +--rw artist [name]
      |  |  +--rw name     string
      |  |  +--rw album [name]
      |  |     +--rw name     string
      |  |     +--rw genre?   identityref
      |  |     +--rw year?    uint16
      |  |     +--rw song [name]
      |  |        +--rw name        string
      |  |        +--rw location    string
      |  |        +--rw format?     string
      |  |        +--rw length?     uint32
      |  +--ro artist-count?   uint32
      |  +--ro album-count?    uint32
      |  +--ro song-count?     uint32
      +--rw playlist [name]
      |  +--rw name           string
      |  +--rw description?   string
      |  +--rw song [index]
      |     +--rw index    uint32
      |     +--rw id       instance-identifier
      +--rw player
         +--rw gap?   decimal64


  rpcs:


   +---x play
      +--ro input
         +--ro playlist       string
         +--ro song-number    uint32
```

C.1. example-jukebox YANG Module

```
   module example-jukebox {

      namespace "http://example.com/ns/example-jukebox";
      prefix "jbox";
      import ietf-restconf { prefix rc; }
```

```
   organization "Example, Inc.";
   contact "support at example.com";
   description "Example Jukebox Data Model Module";
   revision "2013-12-21" {
     description "Initial version.";
     reference "example.com document 1-4673";
   }

   identity genre {
     description "Base for all genre types";
   }

   // abbreviated list of genre classifications
   identity alternative {
     base genre;
     description "Alternative music";
   }
   identity blues {
     base genre;
     description "Blues music";
   }
   identity country {
     base genre;
     description "Country music";
   }
   identity jazz {
     base genre;
     description "Jazz music";
   }
   identity pop {
     base genre;
     description "Pop music";
   }
   identity rock {
     base genre;
     description "Rock music";
   }

   container jukebox {
     presence
       "An empty container indicates that the jukebox
        service is available";

     description
       "Represents a jukebox resource, with a library, playlists,
        and a play operation.";

     container library {
```

```
          description "Represents the jukebox library resource.";

          list artist {
            key name;

            description
              "Represents one artist resource within the
               jukebox library resource.";

            leaf name {
              type string {
                length "1 .. max";
              }
              description "The name of the artist.";
            }

            list album {
              key name;

              description
                "Represents one album resource within one
                 artist resource, within the jukebox library.";

              leaf name {
                type string {
                  length "1 .. max";
                }
                description "The name of the album.";
              }

              leaf genre {
                type identityref { base genre; }
                description
                  "The genre identifying the type of music on
                   the album.";
              }

              leaf year {
                type uint16 {
                  range "1900 .. max";
                }
                description "The year the album was released";
              }

              list song {
                key name;

                description
```

```
                 "Represents one song resource within one
                  album resource, within the jukebox library.";

               leaf name {
                 type string {
                    length "1 .. max";
                 }
                 description "The name of the song";
               }
               leaf location {
                 type string;
                 mandatory true;
                 description
                  "The file location string of the
                   media file for the song";
               }
               leaf format {
                 type string;
                 description
                   "An identifier string for the media type
                    for the file associated with the
                    'location' leaf for this entry.";
               }
               leaf length {
                 type uint32;
                 units "seconds";
                 description
                   "The duration of this song in seconds.";
               }
             }   // end list 'song'
           }   // end list 'album'
         }  // end list 'artist'

         leaf artist-count {
            type uint32;
            units "songs";
            config false;
            description "Number of artists in the library";
         }
         leaf album-count {
            type uint32;
            units "albums";
            config false;
            description "Number of albums in the library";
         }
         leaf song-count {
            type uint32;
            units "songs";
```

```
             config false;
             description "Number of songs in the library";
           }
         }  // end library

         list playlist {
           key name;

           description
             "Example configuration data resource";

           leaf name {
             type string;
             description
               "The name of the playlist.";
           }
           leaf description {
             type string;
             description
               "A comment describing the playlist.";
           }
           list song {
             key index;
             ordered-by user;

             description
               "Example nested configuration data resource";

             leaf index {    // not really needed
               type uint32;
               description
                 "An arbitrary integer index for this
                  playlist song.";
             }
             leaf id {
               type rc:data-resource-identifier;
               mandatory true;
               description
                 "Song identifier. Must identify an instance of
                  /jukebox/library/artist/album/song/name.";
             }
           }
         }

         container player {
           description
             "Represents the jukebox player resource.";
```

```
          leaf gap {
            type decimal64 {
              fraction-digits 1;
              range "0.0 .. 2.0";
            }
            units "tenths of seconds";
            description "Time gap between each song";
          }
        }
      }

      rpc play {
        description "Control function for the jukebox player";
        input {
          leaf playlist {
            type string;
            mandatory true;
            description "playlist name";
          }
          leaf song-number {
            type uint32;
            mandatory true;
            description "Song number in playlist to play";
          }
        }
      }
   }
```

Appendix D. RESTCONF Message Examples

The examples within this document use the normative YANG module defined in Section 7 and the non-normative example YANG module defined in Appendix C.1.

This section shows some typical RESTCONF message exchanges.

D.1. Resource Retrieval Examples

D.1.1. Retrieve the Top-level API Resource

The client may start by retrieving the top-level API resource, using the entry point URI "/restconf".

```
GET /restconf   HTTP/1.1
Host: example.com
Accept: application/yang.api+json
```

The server might respond as follows:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Content-Type: application/yang.api+json

{
  "ietf-restconf:restconf": {
    "data" : [ null ],
    "modules": {
      "module": [
        {
          "name" : "example-jukebox",
          "revision" : "2013-12-21",
          "namespace" : "http://example.com/ns/example-jukebox",
          "schema" : [ null ]
        }
      ]
    },
    "operations" : {
       "play" : [ null ]
    },
    "streams" : {
      "stream" : [
        {
          "name" : "NETCONF",
          "description" : "default NETCONF event stream",
          "replay-support" : true,
          "replay-log-creation-time:" : "2007-07-08T00:00:00Z",
          "events" : [ null ]
        }
      ]
    },
    "version": "1.0"
  }
}
```

To request that the response content to be encoded in XML, the "Accept" header can be used, as in this example request:

```
GET /restconf HTTP/1.1
Host: example.com
Accept: application/yang.api+xml
```

The server will return the same response either way, which might be as follows :

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.api+xml
```


```
<restconf xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">
  <data/>
  <modules>
    <module>
      <name>example-jukebox</name>
      <revision>2013-12-21</revision>
      <namespace>
        http://example.com/ns/example-jukebox
      </namespace>
      <schema />
    </module>
  </modules>
  <operations>
    <play xmlns="http://example.com/ns/example-jukebox"/>
  </operations>
  <streams>
    <stream>
      <name>NETCONF</name>
      <description>default NETCONF event stream
      </description>
      <replay-support>true</replay-support>
      <replay-log-creation-time>
         2007-07-08T00:00:00Z
      </replay-log-creation-time>
      <events/>
    </stream>
  </streams>
  <version>1.0</version>
</restconf>
```


D.1.2. Retrieve The Server Module Information

In this example the client is retrieving the modules resource from the server in JSON format:

```
GET /restconf/modules  HTTP/1.1
Host: example.com
Accept: application/yang.api+json
```

The server might respond as follows.

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Last-Modified: Sun, 22 Apr 2012 01:00:14 GMT
Content-Type: application/yang.api+json
```


```
{
  "ietf-restconf:modules": {
    "module": [
      {
        "name" : "foo",
        "revision" : "2012-01-02",
        "schema" : [null],
        "namespace" : "http://example.com/ns/foo",
        "feature" : [ "feature1", "feature2" ]
      },
      {
        "name" : "foo-types",
        "revision" : "2012-01-05",
        "schema" : [null],
        "namespace" : "http://example.com/ns/foo-types"
      },
      {
        "name" : "bar",
        "revision" : "2012-11-05",
        "schema" : [null],
        "namespace" : "http://example.com/ns/bar",
        "feature" : [ "bar-ext" ],
        "submodule" : [
          {
            "name" : "bar-submod1",
            "revision" : "2012-11-05",
            "schema" : [null]
          },
          {
            "name" : "bar-submod2",
            "revision" : "2012-11-05",
            "schema" : [null]
          }
        ]
      }
    ]
  }
}
```

D.2. Edit Resource Examples

D.2.1. Create New Data Resources

To create a new "artist" resource within the "library" resource, the client might send the following request.

```
POST /restconf/data/example-jukebox:jukebox/library HTTP/1.1
Host: example.com
Content-Type: application/yang.data+json

{ "example-jukebox:artist" : {
    "name" : "Foo Fighters"
  }
}
```

If the resource is created, the server might respond as follows. Note that the "Location" header line is wrapped for display purposes only:

```
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2012 17:02:00 GMT
Server: example-server
Location: http://example.com/restconf/data/
     example-jukebox:jukebox/library/artist/Foo%20Fighters
Last-Modified: Mon, 23 Apr 2012 17:02:00 GMT
ETag: b3830f23a4c
```

To create a new "album" resource for this artist within the "jukebox" resource, the client might send the following request. Note that the request URI header line is wrapped for display purposes only:

```
POST /restconf/data/example-jukebox:jukebox/
   library/artist/Foo%20Fighters  HTTP/1.1
Host: example.com
Content-Type: application/yang.data+json

{
  "example-jukebox:album" : {
    "name" : "Wasting Light",
    "genre" : "example-jukebox:alternative",
    "year" : 2012    # note this is the wrong date
  }
}
```

If the resource is created, the server might respond as follows. Note that the "Location" header line is wrapped for display purposes only:

```
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2012 17:03:00 GMT
Server: example-server
Location: http://example.com/restconf/data/
  example-jukebox:jukebox/library/artist/Foo%20Fighters/
  album/Wasting%20Light
Last-Modified: Mon, 23 Apr 2012 17:03:00 GMT
ETag: b8389233a4c
```

D.2.2. Detect Resource Entity Tag Change

In this example, the server just supports the mandatory datastore last-changed timestamp. The client has previously retrieved the "Last-Modified" header and has some value cached to provide in the following request to patch an "album" list entry with key value "Wasting Light". Only the "year" field is being updated.

```
PATCH /restconf/data/example-jukebox:jukebox/
  library/artist/Foo%20Fighters/album/Wasting%20Light/year
  HTTP/1.1
Host: example.com
Accept: application/yang.data+json
If-Unmodified-Since: Mon, 23 Apr 2012 17:01:00 GMT
Content-Type: application/yang.data+json

{ "example-jukebox:year" : "2011" }
```

In this example the datastore resource has changed since the time specified in the "If-Unmodified-Since" header. The server might respond:

```
HTTP/1.1 412 Precondition Failed
Date: Mon, 23 Apr 2012 19:01:00 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 17:45:00 GMT
ETag: b34aed893a4c
```

D.3. Query String Parameter Examples

D.3.1. "content" Parameter

The "content" parameter is used to select the type of data sub-resources (configuration and/or not configuration) that are returned by the server for a GET method request.

In this example, a simple YANG list that has configuration and non-configuration sub-resources.

```
container events
  list event {
    key name;
    leaf name { type string; }
    leaf description { type string; }
    leaf event-count {
      type uint32;
      config false;
    }
  }
}
```

Example 1: content=all

To retrieve all the sub-resources, the "content" parameter is set to "all". The client might send:

```
GET /restconf/data/example-events:events?content=all
    HTTP/1.1
Host: example.com
Accept: application/yang.data+json
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.data+json

{
  "example-events:events" : {
    "event" : [
      {
        "name" : "interface-up",
        "description" : "Interface up notification count",
        "event-count" : 42
      },
      {
        "name" : "interface-down",
        "description" : "Interface down notification count",
        "event-count" : 4
      }
    ]
  }
}
```

Example 2: content=config

To retrieve only the configuration sub-resources, the "content" parameter is set to "config" or omitted since this is the default value. Note that the "ETag" and "Last-Modified" headers are only returned if the content parameter value is "config".

```
GET /restconf/data/example-events:events?content=config
   HTTP/1.1
Host: example.com
Accept: application/yang.data+json
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:11:30 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
ETag: eeeada438af
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.data+json
```


```
{
  "example-events:events" : {
    "event" : [
      {
        "name" : "interface-up",
        "description" : "Interface up notification count"
      },
      {
        "name" : "interface-down",
        "description" : "Interface down notification count"
      }
    ]
  }
}
```


Example 3: content=non-config

To retrieve only the non-configuration sub-resources, the "content" parameter is set to "non-config". Note that configuration ancestors (if any) and list key leafs (if any) are also returned. The client might send:

```
GET /restconf/data/example-events:events?content=non-config
   HTTP/1.1
Host: example.com
```

```
   Accept: application/yang.data+json
```

The server might respond:

```
   HTTP/1.1 200 OK
   Date: Mon, 23 Apr 2012 17:11:30 GMT
   Server: example-server
   Cache-Control: no-cache
   Pragma: no-cache
   Content-Type: application/yang.data+json

   {
     "example-events:events" : {
       "event" : [
         {
           "name" : "interface-up",
           "event-count" : 42
         },
         {
           "name" : "interface-down",
           "event-count" : 4
         }
       ]
     }
   }
```

D.3.2. "depth" Parameter

The "depth" parameter is used to limit the number of levels of sub-resources that are returned by the server for a GET method request.

This example shows how different values of the "depth" parameter would affect the reply content for retrieval of the top-level "jukebox" data resource.

Example 1: depth=unbounded

To retrieve all the sub-resources, the "depth" parameter is not present or set to the default value "unbounded". Note that some strings are wrapped for display purposes only.

```
   GET /restconf/data/example-jukebox:jukebox?depth=unbounded
      HTTP/1.1
   Host: example.com
   Accept: application/yang.data+json
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.data+json
```


```
{
  "example-jukebox:jukebox" : {
    "library" : {
      "artist" : [
        {
          "name" : "Foo Fighters",
          "album" : [
            {
              "name" : "Wasting Light",
              "genre" : "example-jukebox:alternative",
              "year" : 2011,
              "song" : [
                {
                  "name" : "Wasting Light",
                  "location" :
                    "/media/foo/a7/wasting-light.mp3",
                  "format" : "MP3",
                  "length" " 286
                },
                {
                  "name" : "Rope",
                  "location" : "/media/foo/a7/rope.mp3",
                  "format" : "MP3",
                  "length" " 259
                }
              ]
            }
          ]
        }
      ]
    },
    "playlist" : [
      {
        "name" : "Foo-One",
        "description" : "example playlist 1",
        "song" : [
          {
            "index" : 1,
            "id" : "http://example.com/restconf/data/
                  example-jukebox:jukebox/library/artist/
                  Foo%20Fighters/album/Wasting%20Light/
```

```
                        song/Rope"
                },
                {
                  "index" : 2,
                  "id" : "http://example.com/restconf/data/
                        example-jukebox:jukebox/library/artist/
                        Foo%20Fighters/album/Wasting%20Light/song/
                        Bridge%20Burning"
                }
              ]
            }
          ],
          "player" : {
            "gap" : 0.5
          }
        }
      }
```


Example 2: depth=1

To determine if 1 or more resource instances exist for a given target resource, the value "1" is used.

```
GET /restconf/data/example-jukebox:jukebox?depth=1 HTTP/1.1
Host: example.com
Accept: application/yang.data+json
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.data+json

{
  "example-jukebox:jukebox" : [null]
}
```

Example 3: depth=3

To limit the depth level to the target resource plus 2 sub-resource layers the value "3" is used.

```
GET /restconf/data/example-jukebox:jukebox?depth=3 HTTP/1.1
Host: example.com
Accept: application/yang.data+json
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:11:30 GMT
Server: example-server
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/yang.data+json
```


```
{
  "example-jukebox:jukebox" : {
    "library" : {
      "artist" : [ null ]
    },
    "playlist" : [
      {
        "name" : "Foo-One",
        "description" : "example playlist 1",
        "song" : [ null ]
      }
    ],
    "player" : {
      "gap" : 0.5
    }
  }
}
```


D.3.3. "filter" Parameter

The following URIs show some examples of notification filter specifications (lines wrapped for display purposes only):

```
// filter = /event/eventClass='fault'
GET /restconf/streams/stream/NETCONF/events?
   filter=%2Fevent%2FeventClass%3D'fault'

// filter = /event/severityCode<=4
GET /restconf/streams/stream/NETCONF/events?
   filter=%2Fevent%2FseverityCode%3C%3D4

// filter = /linkUp|/linkDown
GET /restconf/streams/stream/SNMP/events?
   filter=%2FlinkUp%7C%2FlinkDown

// filter = /*/reportingEntity/card!='Ethernet0'
GET /restconf/streams/stream/NETCONF/events?
   filter=%2F*%2FreportingEntity%2Fcard%21%3D'Ethernet0'

// filter = /*/email-addr[contains(.,'company.com')]
GET /restconf/streams/stream/critical-syslog/events?
   filter=%2F*%2Femail-addr[contains(.%2C'company.com')]

// Note: the module name is used as prefix.
// filter = (/example-mod:event1/name='joe' and
//           /example-mod:event1/status='online')
GET /restconf/streams/stream/NETCONF/events?
  filter=(%2Fexample-mod%3Aevent1%2Fname%3D'joe'%20and
          %20%2Fexample-mod%3Aevent1%2Fstatus%3D'online')
```

D.3.4. "insert" Parameter

In this example, a new first entry in the "Foo-One" playlist is being created.

Request from client:

```
POST /restconf/data/example-jukebox:jukebox/
  playlist/Foo-One?insert=first HTTP/1.1
Host: example.com
Content-Type: application/yang.data+json

{
  "example-jukebox:song" : {
     "index" : 1,
     "id" : "/example-jukebox:jukebox/library/artist/
         Foo%20Fighters/album/Wasting%20Light/song/Rope"
   }
}
```

Response from server:

```
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2012 13:01:20 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
Location: http://example.com/restconf/data/
   example-jukebox:jukebox/playlist/Foo-One/song/1
ETag: eeeada438af
```

D.3.5. "point" Parameter

Example:

In this example, the client is inserting a new "song" resource within an "album" resource after another song. The request URI is split for display purposes only.

Request from client:

```
POST /restconf/data/example-jukebox:jukebox/
   library/artist/Foo%20Fighters/album/Wasting%20Light?
   insert=after&point=%2Fexample-jukebox%3Ajukebox%2F
   library%2Fartist%2FFoo%20Fighters%2Falbum%2F
   Wasting%20Light%2Fsong%2FBridge%20Burning   HTTP/1.1
Host: example.com
Content-Type: application/yang.data+json

{
  "example-jukebox:song" : {
     "name" : "Rope",
     "location" : "/media/foo/a7/rope.mp3",
     "format" : "MP3",
     "length" : 259
  }
}
```

Response from server:

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 13:01:20 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
ETag: abcada438af
```

D.3.6. "select" Parameter

TBD

D.3.7. "start-time" Parameter

TBD

D.3.8. "stop-time" Parameter

TBD

Authors' Addresses

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Kent Watsen
Juniper Networks

Email: kwatsen@juniper.net

Rex Fernando
Cisco

Email: rex@cisco.com

Network Working Group A. Bierman
Internet-Draft YumaWorks
Intended status: Standards Track M. Bjorklund
Expires: June 24, 2014 Tail-f Systems
K. Watsen
Juniper Networks
R. Fernando
Cisco
December 21, 2013

YANG Patch Media Type
draft-bierman-netconf-yang-patch-00

Abstract

This document describes a method for applying patches to NETCONF datastores using data defined with the YANG data modeling language.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

There is a need for standard mechanisms to patch NETCONF [RFC6241] datastores which contain conceptual data that conforms to schema specified with YANG [RFC6020]. An "ordered edit list" approach is needed to provide client developers with a simpler edit request format that can be more efficient and also allow more precise client control of the transaction procedure than existing mechanisms.

This document defines a media type for a YANG-based editing mechanism that can be used with the HTTP PATCH method [RFC5789] or custom NETCONF operations (defined with the YANG rpc-stmt).

YANG Patch is designed to support multiple protocols with the same mechanisms. The RESTCONF [RESTCONF] protocol utilizes YANG Patch with the HTTP PATCH method. A new RPC operation can be defined to utilize YANG Patch in the NETCONF protocol. Both the RESTCONF and NETCONF protocols are designed to utilize the YANG data modeling language to specify content schema modules.

1.1. Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

1.1.1. NETCONF

The following terms are defined in [RFC6241]:

- candidate configuration datastore
- client
- configuration data
- datastore
- configuration datastore
- protocol operation
- running configuration datastore
- server

- startup configuration datastore
- state data
- user

1.1.2. HTTP

The following terms are defined in [RFC2616]:

- entity tag
- fragment
- header line
- message body
- method
- path
- query
- request URI
- response body

1.1.3. YANG

The following terms are defined in [RFC6020]:

- container
- data node
- key leaf
- leaf
- leaf-list
- list
- presence container (or P-container)
- RPC operation (now called protocol operation)

- o non-presence container (or NP-container)
- o ordered-by system
- o ordered-by user

1.1.4. Terms

The following terms are used within this document:

- o YANG Patch: a conceptual edit request using the "yang-patch" YANG container, defined in Section 3. In HTTP, refers to a PATCH method where the media type is "application/yang.patch+xml" or "application/yang.patch+json".
- o YANG Patch Status: a conceptual edit status response using the YANG "yang-patch-status" container, defined in Section 3. In HTTP, refers to a response message for a PATCH method, where the message body is identified by the media type "application/yang.patch-status+xml" or "application/yang.patch-status+json".

1.1.5. Tree Diagrams

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

- o Brackets "[" and "]" enclose list keys.
- o Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- o Symbols after data node names: "?" means an optional node 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. YANG Patch

A "YANG Patch" is an ordered list of edits that are applied to the target datastore by the server. The specific fields are defined with the 'yang-patch' container definition in the YANG module Section 3.

For RESTCONF, the YANG Patch operation is invoked by the client by sending a PATCH method request with the YANG Patch media type. A message body representing the YANG Patch input parameters MUST be provided.

For NETCONF, a YANG "rpc" statement must be defined. The "yang-patch" grouping MUST be included in the input parameters and the "yang-patch-status" grouping MUST be included in the output parameters.

2.1. Target Resource

The YANG Patch operation uses a conceptual root within a NETCONF configuration datastore to identity the patch point for the edit operation. This root can be the datastore itself, or 1 or more data nodes within the datastore.

For RESTCONF, the target resource is derived from the request URI.

For NETCONF, the target resource MUST be defined as an input parameter in the YANG "rpc" statement.

2.2. yang-patch Input

A data element representing the YANG Patch is sent by the client to specify the edit operation request. When used with the HTTP PATCH method, this data is identified by the YANG Patch media type.

YANG Tree Diagram For "yang-patch" Container

```
+--rw yang-patch
   +--rw patch-id?   string
   +--rw comment?    string
   +--rw edit [edit-id]
      +--rw edit-id      string
      +--rw operation    enumeration
      +--rw target       target-resource-offset
      +--rw point?       target-resource-offset
      +--rw where?       enumeration
      +--rw value
```

2.3. yang-patch-status Output

A data element representing the YANG Patch Status is returned to the client to report the detailed status of the edit operation. When used with the HTTP PATCH method, this data is identified by the YANG Patch Status media type.

YANG Tree Diagram For "yang-patch-status" Container:

```
+--rw yang-patch-status
   +--rw patch-id?        string
   +--rw (global-status)?
   |  +--:(global-errors)
   |  |  +--ro errors
   |  |
   |  +--:(ok)
   |     +--rw ok?              empty
   +--rw edit-status
      +--rw edit [edit-id]
         +--rw edit-id     string
         +--rw (edit-status-choice)?
            +--:(ok)
            |  +--rw ok?         empty
            +--:(location)
            |  +--rw location?   inet:uri
            +--:(errors)
               +--ro errors
```

2.4. Target Data Node

The target data node for each edit operation is determined by the value of the target resource in the request and the "target" leaf within each "edit" entry.

If the target resource specified in the request URI identifies a datastore resource, then the path string in the "target" leaf is an absolute path expression. The first node specified in the "target" leaf is a top-level data node defined within a YANG module.

If the target resource specified in the request URI identifies a data resource, then the path string in the "target" leaf is a relative path expression. The first node specified in the "target" leaf is a child node of the data node associated with the target resource.

2.5. Edit Operations

Each YANG patch edit specifies one edit operation on the target data node. The set of operations is aligned with the NETCONF edit operations, but also includes some new operations.

Operation	Description
create	create a new data resource if it does not already exist or error
delete	delete a data resource if it already exists or error
insert	insert a new user-ordered data resource
merge	merge the edit value with the target data resource; create if it does not already exist
move	re-order the target data resource
replace	replace the target data resource with the edit value
remove	remove a data resource if it already exists or no error

YANG Patch Edit Operations

2.6. Error Handling

If a well-formed, schema-valid YANG Patch message is received, then then the server will process the supplied edits in ascending order. The following error modes apply to the processing of this edit list:

All the specified edits MUST be applied or the target datastore contents SHOULD be returned to its original state before the PATCH method started. The server MAY fail to restore the contents of the target datastore completely and with certainty. It is possible for a rollback to fail or an "undo" operation to fail.

The server will save the running datastore to non-volatile storage if it has changed, after the edits have been attempted.

3. YANG Module

The "ietf-yang-patch" module defines conceptual definitions within groupings, which are not meant to be implemented as datastore contents by a server.

The "ietf-yang-types" and "ietf-inet_types" modules from [RFC6991] are used by this module for some type definitions.

The "ietf-restconf" module from [RESTCONF] is used by this module for a grouping definition.

RFC Ed.: update the date below with the date of RFC publication and remove this note.

<CODE BEGINS> file "ietf-yang-patch@2013-12-21.yang"

```
module ietf-yang-patch {
  namespace "urn:ietf:params:xml:ns:yang:ietf-yang-patch";
  prefix "ypatch";

  import ietf-inet-types { prefix inet; }
  import ietf-restconf { prefix rc; }

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

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

     WG Chair: Bert Wijnen
               <mailto:bertietf@bwijnen.net>

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

     Editor:   Andy Bierman
               <mailto:andy@yumaworks.com>

     Editor:   Martin Bjorklund
               <mailto:mbj@tail-f.com>

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

     Editor:   Rex Fernando
               <mailto:rex@cisco.com>";
```

```
     description
       "This module contains conceptual YANG specifications
        for the YANG Patch and YANG Patch Status data structures.

        Note that the YANG definitions within this module do not
        represent configuration data of any kind.
        The YANG grouping statements provide a normative syntax
        for XML and JSON message encoding purposes.

        Copyright (c) 2013 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.";

     // RFC Ed.: replace XXXX with actual RFC number and remove this
     // note.

     // RFC Ed.: remove this note
     // Note: extracted from draft-bierman-netconf-yang-patch-00.txt

     // RFC Ed.: update the date below with the date of RFC publication
     // and remove this note.
     revision 2013-12-21 {
       description
         "Initial revision.";
       reference
         "RFC XXXX: YANG Patch";
     }

     typedef target-resource-offset {
       type string {
         length "1 .. max";
       }
       description
         "Contains a relative Data Resource Identifier formatted string
          to identify a specific data sub-resource instance.
          The document root for all data resources is a
          target data resource that is specified in the
          object definition using this data type.";
     }
```

```
     grouping yang-patch {

       description
         "A grouping that contains a YANG container
          representing the syntax and semantics of a
          YANG Patch edit request message.";

       container yang-patch {
         description
           "Represents a conceptual sequence of datastore edits,
            called a patch. Each patch is given a client-assigned
            patch identifier. Each edit MUST be applied
            in ascending order, and all edits MUST be applied.
            If any errors occur, then the target datastore MUST NOT
            be changed by the patch operation.

            A patch MUST be validated by the server to be a
            well-formed message before any of the patch edits
            are validated or attempted.

            YANG datastore validation (defined in RFC 6020, section
            8.3.3) is performed after all edits have been
            individually validated.

            It is possible for a datastore constraint violation to occur
            due to any node in the datastore, including nodes not
            included in the edit list. Any validation errors MUST
            be reported in the reply message.";

         reference
           "RFC 6020, section 8.3.";

         leaf patch-id {
           type string;
           description
             "An arbitrary string provided by the client to identify
              the entire patch.  This value SHOULD be present in any
              audit logging records generated by the server for the
              patch. Error messages returned by the server pertaining
              to this patch will be identified by this patch-id value.";
         }

         leaf comment {
           type string {
             length "0 .. 1024";
           }
           description
             "An arbitrary string provided by the client to describe
```

```
             the entire patch.  This value SHOULD be present in any
             audit logging records generated by the server for the
             patch.";
        }

        list edit {
          key edit-id;
          ordered-by user;

          description
            "Represents one edit within the YANG Patch
             request message.";

          leaf edit-id {
            type string;
            description
              "Arbitrary string index for the edit.
               Error messages returned by the server pertaining
               to a specific edit will be identified by this
               value.";
          }

          leaf operation {
            type enumeration {
              enum create {
                description
                  "The target data node is created using the
                   supplied value, only if it does not already
                   exist.";
              }
              enum delete {
                description
                  "Delete the target node, only if the data resource
                   currently exists, otherwise return an error.";
              }
              enum insert {
                description
                  "Insert the supplied value into a user-ordered
                   list or leaf-list entry. The target node must
                   represent a new data resource.";
              }
              enum merge {
                description
                  "The supplied value is merged with the target data
                   node.";
              }
              enum move {
                description
```

```
                "Move the target node. Reorder a user-ordered
                 list or leaf-list. The target node must represent
                 an existing data resource.";
            }
            enum replace {
              description
                "The supplied value is used to replace the target
                 data node.";
            }
            enum remove {
              description
                "Delete the target node if it currently exists.";
            }
          }
          mandatory true;
          description
            "The datastore operation requested for the associated
             edit entry";
        }

        leaf target {
          type target-resource-offset;
          mandatory true;
          description
            "Identifies the target data resource for the edit
             operation.";
        }

        leaf point {
          when "(../operation = 'insert' or " +
            "../operation = 'move') and " +
            "(../where = 'before' or ../where = 'after')" {
            description
              "Point leaf only applies for insert or move
               operations, before or after an existing entry.";
          }
          type target-resource-offset;
          description
            "The absolute URL path for the data node that is being
             used as the insertion point or move point for the
             target of this edit entry.";
        }

        leaf where {
          when "../operation = 'insert' or ../operation = 'move'" {
            description
              "Where leaf only applies for insert or move
               operations.";
```

```
             }
             type enumeration {
               enum before {
                 description
                   "Insert or move a data node before the data resource
                    identified by the 'point' parameter.";
               }
               enum after {
                 description
                   "Insert or move a data node after the data resource
                    identified by the 'point' parameter.";
               }
               enum first {
                 description
                   "Insert or move a data node so it becomes ordered
                    as the first entry.";
               }
               enum last {
                 description
                   "Insert or move a data node so it becomes ordered
                    as the last entry.";
               }

             }
             default last;
             description
               "Identifies where a data resource will be inserted or
                moved. YANG only allows these operations for
                list and leaf-list data nodes that are ordered-by
                user.";
           }

           anyxml value {
             when "(../operation = 'create' or " +
               "../operation = 'merge' " +
               "or ../operation = 'replace' or " +
               "../operation = 'insert')" {
               description
                 "Value node only used for create, merge,
                  replace, and insert operations";
             }
             description
               "Value used for this edit operation.";
           }
         }
       }

     } // grouping yang-patch
```

```
     grouping yang-patch-status {

       description
         "A grouping that contains a YANG container
          representing the syntax and semantics of
          YANG Patch status response message.";

       container yang-patch-status {
         description
           "A container representing the response message
            sent by the server after a YANG Patch edit
            request message has been processed.";

         leaf patch-id {
           type string;
           description
             "The patch-id value used in the request";
         }

         choice global-status {
           description
             "Report global errors or complete success.
              If there is no case selected then errors
              are reported in the edit-status container.";

           case global-errors {
             uses rc:errors;
             description
               "This container will be present if global
                errors unrelated to a specific edit occurred.";
           }
           leaf ok {
             type empty;
             description
               "This leaf will be present if the request succeeded
                and there are no errors reported in the edit-status
                container.";
           }
         }

         container edit-status {
           description
             "This container will be present if there are
              edit-specific status responses to report.";

           list edit {
             key edit-id;
```

```
            description
              "Represents a list of status responses,
               corresponding to edits in the YANG Patch
               request message.  If an edit entry was
               skipped or not reached by the server,
               then this list will not contain a corresponding
               entry for that edit.";

            leaf edit-id {
              type string;
               description
                 "Response status is for the edit list entry
                  with this edit-id value.";
            }
            choice edit-status-choice {
              description
                "A choice between different types of status
                 responses for each edit entry.";
              leaf ok {
                type empty;
                description
                  "This edit entry was invoked without any
                   errors detected by the server associated
                   with this edit.";
              }
              leaf location {
                type inet:uri;
                description
                  "Contains the Location header value that would be
                   returned if this edit causes a new resource to be
                   created. If the edit identified by the same edit-id
                   value was successfully invoked and a new resource
                   was created, then this field will be returned
                   instead of 'ok'.";
              }
              case errors {
                uses rc:errors;
                description
                  "The server detected errors associated with the
                     edit identified by the same edit-id value.";
              }
            }
          }
        }
      }
    }  // grouping yang-patch-status

  }
```

<CODE ENDS>

4. IANA Considerations

4.1. YANG Module Registry

This document registers one URI in the IETF XML registry [RFC3688]. Following the format in RFC 3688, the following registration is requested to be made.

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

This document registers one YANG module in the YANG Module Names registry [RFC6020].

```
name:         ietf-yang-patch
namespace:    urn:ietf:params:xml:ns:yang:ietf-yang-patch
prefix:       ypatch
// RFC Ed.: replace XXXX with RFC number and remove this note
reference:    RFC XXXX
```

4.2. application/yang.patch Media Types

The MIME media type for a YANG Patch document is application/ yang.patch.

Type name: application

Subtype name: yang.patch

Required parameters: TBD

Optional parameters: TBD

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

// RFC Ed.: replace XXXX with RFC number and remove this note
Published specification: RFC XXXX

4.3. application/yang.patch-status Media Types

The MIME media type for a YANG Patch status document is application/ yang.patch-status.

Type name: application

Subtype name: yang.patch-status

Required parameters: TBD

Optional parameters: TBD

Encoding considerations: TBD

Security considerations: TBD

Interoperability considerations: TBD

// RFC Ed.: replace XXXX with RFC number and remove this note
Published specification: RFC XXXX

5. Security Considerations

TBD

6. Normative References

[JSON] Bray, T., Ed., "The JSON Data Interchange Format", draft-ietf-json-rfc4627bis-10 (work in progress), December 2013.

[RESTCONF] Bierman, A., Bjorklund, M., Watsen, K., and R. Fernando, "RESTCONF Protocol", draft-bierman-netconf-restconf-03 (work in progress), December 2013.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, March 2010.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013.

[W3C.REC-xml-20081126] Yergeau, F., Maler, E., Paoli, J., Sperberg-McQueen, C., and T. Bray, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126>.

Appendix A. Change Log

-- RFC Ed.: remove this section before publication.

A.1. RESTCONF-02 to YANG-PATCH-00

- o Split out from RESTCONF draft
- o Rewrite HTTP sections to be more generic and apply to NETCONF as well

Appendix B. Example YANG Module

The example YANG module used in this document represents a simple media jukebox interface. The "example-jukebox" YANG module is defined in [RESTCONF].

YANG Tree Diagram for "example-jukebox" Module:

```
+--rw jukebox?
   +--rw library
   |  +--rw artist [name]
   |  |  +--rw name     string
   |  |  +--rw album [name]
   |  |     +--rw name     string
   |  |     +--rw genre?   identityref
   |  |     +--rw year?    uint16
   |  |     +--rw song [name]
   |  |        +--rw name        string
   |  |        +--rw location    string
   |  |        +--rw format?     string
   |  |        +--rw length?     uint32
   |  +--ro artist-count?   uint32
   |  +--ro album-count?    uint32
   |  +--rw song-count?     uint32
   +--rw playlist [name]
   |  +--rw name           string
   |  +--rw description?   string
   |  +--rw song [index]
   |     +--rw index    uint32
   |     +--rw id       instance-identifier
   +--rw player
      +--rw gap?   decimal64


rpcs:


+---x play
   +--ro input
      +--ro playlist       string
      +--ro song-number    uint32
```

B.1. YANG Patch Examples

This section includes RESTCONF examples. NETCONF examples are TBD. Most examples are shown in JSON encoding [JSON], and some are shown in XML encoding [W3C.REC-xml-20081126].

B.1.1. Patch an Existing Data Resource

To replace just the "year" field in the "album" resource (instead of replacing the entire resource), the client might send a plain patch as follows. Note that the request URI header line is wrapped for display purposes only:

```
PATCH /restconf/data/example-jukebox:jukebox/
   library/artist/Foo%20Fighters/album/Wasting%20Light HTTP/1.1
Host: example.com
If-Match: b8389233a4c
Content-Type: application/yang.data+json

{ "example-jukebox:year" : 2011 }
```

If the field is updated, the server might respond:

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:49:30 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 17:49:30 GMT
ETag: b2788923da4c
```

The XML encoding for the same request might be:

```
PATCH /restconf/data/example-jukebox:jukebox/
   library/artist/Foo%20Fighters/album/Wasting%20Light HTTP/1.1
Host: example.com
If-Match: b8389233a4c
Content-Type: application/yang.data+xml


<year xmlns="http://example.com/ns/example-jukebox">2011</year>
```

B.1.2. Add Resources: Error

The following example shows several songs being added to an existing album. Each edit contains one song. The first song already exists, so an error will be reported for that edit. The rest of the edits were not attempted, since the first edit failed.

Request from client:

```
PATCH /restconf/data/example-jukebox:jukebox/
   library/artist/Foo%20Fighters/album/Wasting%20Light HTTP/1.1
Host: example.com
Accept: application/yang.patch-status+json
Content-Type: application/yang.patch+json
```

```
{
  "ietf-restconf:yang-patch" : {
    "patch-id" : "add-songs-patch",
    "edit" : [
      {
        "edit-id" : 1,
        "operation" : "create",
        "target" : "/song",
        "value" : {
          "song" : {
            "name" : "Bridge Burning",
            "location" : "/media/bridge_burning.mp3",
            "format" : "MP3",
            "length" : 288
          }
        }
      },
      {
        "edit-id" : 2,
        "operation" : "create",
        "target" : "/song",
        "value" : {
          "song" : {
            "name" : "Rope",
            "location" : "/media/rope.mp3",
            "format" : "MP3",
            "length" : 259
          }
        }
      },
      {
        "edit-id" : 3,
        "operation" : "create",
        "target" : "/song",
        "value" : {
          "song" : {
            "name" : "Dear Rosemary",
            "location" : "/media/dear_rosemary.mp3",
            "format" : "MP3",
            "length" : 269
          }
        }
      }
    ]
  }
}
```


Response from server:

```
HTTP/1.1 409 Conflict
Date: Mon, 23 Apr 2012 13:01:20 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
Content-Type: application/yang.patch-status+json
```


```
{
  "ietf-restconf:yang-patch-status" : {
    "patch-id" : "add-songs-patch",
    "edit-status" : {
      "edit" : [
        {
          "edit-id" : 1,
          "errors" : {
            "error" : [
              {
                "error-type": "application",
                "error-tag": "data-exists",
                "error-path": "/example-jukebox:jukebox/library
                   /artist/Foo%20Fighters/album/Wasting%20Light
                   /song/Burning%20Light",
                "error-message":
                  "Data already exists, cannot be created"
              }
            ]
          }
        }
      ]
    }
  }
}
```


B.1.3. Add Resources: Success

The following example shows several songs being added to an existing album.

- o Each of 2 edits contains one song.

- o Both edits succeed and new sub-resources are created

Request from client:

```
PATCH /restconf/data/example-jukebox:jukebox/
   library/artist/Foo%20Fighters/album/Wasting%20Light
   HTTP/1.1
Host: example.com
```

```
   Accept: application/yang.patch-status+json
   Content-Type: application/yang.patch+json

   {
     "ietf-restconf:yang-patch" : {
       "patch-id" : "add-songs-patch-2",
       "edit" : [
         {
           "edit-id" : 1,
           "operation" : "create",
           "target" : "/song",
           "value" : {
             "song" : {
               "name" : "Rope",
               "location" : "/media/rope.mp3",
               "format" : "MP3",
               "length" : 259
             }
           }
         },
         {
           "edit-id" : 2,
           "operation" : "create",
           "target" : "/song",
           "value" : {
             "song" : {
               "name" : "Dear Rosemary",
               "location" : "/media/dear_rosemary.mp3",
               "format" : "MP3",
               "length" : 269
             }
           }
         }
       ]
     }
   }
```

Response from server:

```
   HTTP/1.1 409 Conflict
   Date: Mon, 23 Apr 2012 13:01:20 GMT
   Server: example-server
   Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
   Content-Type: application/yang.patch-status+json

   {
     "ietf-restconf:yang-patch-status" : {
       "patch-id" : "add-songs-patch-2",
```

```
          "ok" : [null],
          "edit-status" : {
            "edit" : [
              {
                "edit-id" : 1,
                "location" : "http://example.com/restconf/
                   data/example-jukebox:jukebox/library/artist/
                   Foo%20Fighters/album/Wasting%20Light/song/Rope"
              },
              {
                "edit-id" : 2,
                "location" : "http://example.com/restconf/
                   data/example-jukebox:jukebox/library/artist/
                   Foo%20Fighters/album/Wasting%20Light/song/
                   Dear%20Rosemary"
              }
            ]
          }
        }
      }
```


B.1.4. Move list entry example

The following example shows a song being moved within an existing playlist. Song "1" in playlist "Foo-One" is being moved after song "3" in the playlist. The operation succeeds, so a non-error reply example can be shown.

Request from client:

```
PATCH /restconf/data/example-jukebox:jukebox/
  playlist/Foo-One   HTTP/1.1
Host: example.com
Accept: application/yang.patch-status+json
Content-Type: application/yang.patch+json
```


```
{
  "ietf-restconf:yang-patch" : {
    "patch-id" : "move-song-patch",
    "comment" : "Move song 1 after song 3",
    "edit" : [
      {
        "edit-id" : 1,
        "operation" : "move",
        "target" : "/song/1",
        "point" : "/song3",
        "where" : "after"
      }
    ]
  }
}
```


Response from server:

```
HTTP/1.1 400 OK
Date: Mon, 23 Apr 2012 13:01:20 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 13:01:20 GMT
Content-Type: application/yang.patch-status+json
```


```
{
  "ietf-restconf:yang-patch-status" : {
    "patch-id" : "move-song-patch",
    "ok" : [null],
    "edit-status" : {
      "edit" : [
        {
          "edit-id" : 1,
          "ok" : [ null ]
        }
      ]
    }
  }
}
```

Authors' Addresses

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Kent Watsen
Juniper Networks

Email: kwatsen@juniper.net

Rex Fernando
Cisco

Email: rex@cisco.com

Network Configuration WG
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2014
I. Farrer
Deutsche Telekom AG
M. Abrahamsson
T-Systems
October 20, 2013

NETCONF DHCPv6 Option
draft-fa-netconf-dhcpv6-option-00

Abstract

This document defines DHCPv6 options for bootstrapping the NETCONF protocol on devices.

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

Copyright Notice

Table of Contents

1. Introduction

NETCONF [RFC6241] combined with the YANG [RFC6020] data modeling language provides an extensible mechanism for configuring and monitoring networked devices. One of the advantages that NETCONF/YANG offers over other network management protocols is that it is flexible enough to be adapted for use with almost any service on any device.

The Dynamic Host Configuration Protocol for IPv6 [RFC3315] is widely in use for the configuration of network devices. This document describes a DHCPv6 option which can be used for provisioning the necessary parameters to bootstrap NETCONF connectivity so that a device can then obtain further configuration. An example device suitable for this type of configuration process would be a managed home gateway router.

This document uses the terms "client" and "server" to describe the hosts at either end of a NETCONF connection. "Client" should be understood to be the host that is actively initiating the NETCONF connection to the "Server". These definitions are used to align with the terminology in the DHCPv6 message flow.

2. NETCONF DHCPv6 Container Option

The following section describes the format of the NETCONF configuration container option. A container approach has been taken so that different NETCONF transport protocols can be supported. Currently only two transport protocols have been defined, NETCONF over SSH [RFC6242] and NETCONF over TLS [RFC5539]. If necessary in the future, the option could be extended to support additional transport protocols through the definition of new sub-options.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    OPTION_NETCONF_CONT        |           option-len          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           sub-options                         |
|                           (variable)                          |
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

option-code OPTION_NETCONF_CONT (TBA1)
option-len Variable
sub-options Contains one or more NETCONF configuration sub-options
            (described below).

2.1. NETCONF over SSH Sub-Option

Clients which implement NETCONF transport over Secure Shell (SSH) use the following sub-option to obtain configuration necessary to establish a connection.

The procedure for establishing NETCONF connectivity over SSH, is described in [RFC6242].

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     OPTION_NETCONF_SSH        |          option-len           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    priority   |             dest-port         |               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+               |
|                          server fqdn(s)                       |
|                        (variable length)                      |
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

option-code   OPTION_NETCONF_SSH (TBA2)
option-len   Variable
priority   8-bit integer. Described in Section 3.1
dest-port   16-bit integer to be used by the client as the
                    destination layer 4 port to initiate the SSH
                    connection to.
server-fqdn   List of FQDNs to use for the NETCONF server,
                    formatted according to Section 8 of [RFC3315].

In case the client receives more than one server address in the server-fqdn field, the client SHOULD initiate connections to the addresses in the order they are listed in the server-fqdn field, attempting to establish a connection to each server until one is successfully established.

2.2. NETCONF over TLS Sub-Option

Clients which implement NETCONF transport over Transport Layer Security (TLS) use the following sub-option to obtain configuration necessary to establish a connection.

The procedure for establishing NETCONF connectivity over TLS, is described in [RFC5539].

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   OPTION_NETCONF_TLS          |         option-len            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    priority   |            dest-port          |               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+               |
|                        server-fqdn(s)                         |
|                       (variable length)                       |
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
option-code OPTION_NETCONF_TLS (TBA3)
option-len  Variable
priority    8-bit integer.  Described in Section 3.1
dest-port   16-bit integer to be used by the client as the
            destination layer 4 port for initiating the TLS
            connection
server-fqdn List of FQDNs to use for the NETCONF server, formatted
            according to Section 8 of [RFC3315].
```

In case the client receives more than one server FQDN in the server-fqdn field, the client SHOULD initiate connections to the addresses in the order they are listed in the server-fqdn field, attempting to establish a connection to each server until one is successfully established.

3. DHCPv6 Client Behavior

When a device which implements both NETCONF functionality and the DHCP option described in this document creates a DHCPv6 SOLICIT message, it SHOULD include OPTION_NETCONF_CONT (TBD) within the ORO field.

On receipt of an DHCP ADVERTISE response message including the OPTION_NETCONF_CONT option, the client evaluates the sub-options which it contains as follows:

o If OPTION_NETCONF_CONT does not contain a transport sub-option implemented by the client, then it MUST be discarded by the client.

- o If OPTION_NETCONF_CONT contains a single NETCONF transport protocol sub-option implemented by the client, then the client SHOULD attempt establish a NETCONF session using the configured transport protocol.
- o If OPTION_NETCONF_CONT contains multiple NETCONF transport protocol sub-options supported by the client, then the client SHOULD follow the procedure described below to establish a connection to the NETCONF server.

3.1. Priorities

As NETCONF is not limited to on specific transport protocol, the NETCONF client and/or server may have been deployed with support for more than one NETCONF transport protocol.

The 'priority' field contained within the transport protocol specific sub-options give the service provider a method of indicating to the client the order in which to attempt using the different supported protocols to establish NETCONF connectivity.

A client which supports two (or more) NETCONF transport protocols, and receives configuration parameters for at least two protocols SHOULD inspect the values of the priority field. The sub-option with the highest priority value SHOULD be used as the first NETCONF protocol to attempt for establishing connectivity.

In the event that this connection attempt is not successful, then the client SHOULD attempt to establish connectivity using the NETCONF transport protocol sub-option with the second highest priority, then the third highest priority, and so on until either a successful connection has been established, or there are no more

In the event that the client receives two options with the same priority, the client SHOULD implement a mechanism for prioritising one mechanism over the other. This mechanism is implementation specific.

4. DHCPv6 Server Behavior

When a DHCPv6 server receives a client SOLICIT message containing the OPTION_NETCONF_CONT option code within the ORO field, it SHOULD respond with an ADVERTISE message containing the sub-options

If the operator has deployed their NETCONF infrastructure with support for more than one NETCONF transport protocol and has a preference for clients to use one transport protocol over another, then the 'priorities' field SHOULD be used within the NETCONF transport protocol sub-options to indicate to the client the order to attempt using the protocols for connectivity as described in Section 3.1.

5. Security Considerations

The NETCONF protocol relies on the underlying transport protocol to provide security. Security considerations described in [RFC5539] and [RFC6242] are also applicable to this document.

6. IANA Considerations

IANA is kindly asked to allocate DHCPv6 option codes for OPTION_NETCONF_CONT, OPTION_NETCONF_SSH and OPTION_NETCONF_TLS.

7. Acknowledgements

Many thanks to everyone.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3315] Droms, R., Bound, J., Volz, B., Lemon, T., Perkins, C. and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, July 2003.

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J. and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

8.2. Informative References

[RFC5539] Badra, M., "NETCONF over Transport Layer Security (TLS)", RFC 5539, May 2009.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.

Index

Authors' Addresses

Ian Farrer
Deutsche Telekom AG
Bonn,
Germany

Email: ian.farrer@telekom.de

Mikael Abrahamsson
T-Systems
Stockholm,
Sweden

Email: mikael.abrahamsson@t-systems.se

NETCONF Working Group K. Watsen
Internet-Draft Juniper Networks
Updates: 4253 (if approved) July 2014
Intended status: Standards Track
Expires: January 02, 2015

NETCONF Call Home using SSH
draft-ietf-netconf-reverse-ssh-06

Abstract

This document presents a technique for a NETCONF server to request that a NETCONF client initiates a SSH connection to the NETCONF server, a technique referred to as 'call home'. Call home is needed to support deployments where the NETCONF client is otherwise unable to initiate a SSH connection to the NETCONF server directly.

Status of This Memo

Copyright Notice

Table of Contents

1. 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. Introduction

This document presents a technique for a NETCONF server to request that a NETCONF [RFC6241] client initiates a SSH [RFC4251] connection to the NETCONF server, a technique referred to as 'call home'. Call home is needed to support deployments where the NETCONF client is otherwise unable to initiate a SSH connection to the NETCONF server directly.

2.1. Applicability Statement

The techniques described in this document are suitable for network management scenarios such as the ones described in section 3. However, these techniques SHOULD only be used for a NETCONF server to initiate a connection to a NETCONF client, as described in this document.

The reason for this restriction is that different protocols have different security assumptions. The NETCONF over SSH specification requires NETCONF clients and servers to verify the identity of the other party before starting the NETCONF protocol (section 6 of [RFC6242]). This contrasts with the base SSH protocol, which does not require programmatic verification of the other party (section 9.3.4 of [RFC4251] and section 4 of [RFC4252]). In such circumstances, allowing the SSH server to contact the SSH client would open new vulnerabilities. Therefore, any use of call home with SSH for purposes other than NETCONF will need a thorough, contextual security analysis.

2.2. Update to RFC 4253

This document updates the SSH Transport Layer Protocol [RFC4253] only by removing the restriction in Section 4 (Connection Setup) of [RFC4252] that the SSH Client must initiate the transport connection. Security implications related to this change are discussed in Security Considerations (Section 7).

2.3. Draft Naming

(this section should be removed if this draft becomes an RFC)

This draft's name includes the string "reverse-ssh", and yet currently nowhere in this draft is there any reference to reversing SSH. This appearant ommision comes from the -05 edit of this draft, where "Reverse SSH" was changed to "Call Home" throughout. If this draft becomes an RFC, its name would no longer contain the obsolete "reverse-ssh" reference, thus self-correcting this inconsistency.

3. Benefits to Device Management

The SSH protocol is nearly ubiquitous for device management, as it is the transport for the command-line applications ‘ssh‘, ‘scp‘, and ‘sftp‘ and is the required transport for the NETCONF protocol [RFC6241]. However, all these SSH-based protocols expect the network element to be the SSH server.

NETCONF over SSH Call Home enables the network element to consistently be the SSH server regardless of which peer initiates the underlying TCP connection. Maintaining the role of SSH server is both necessary and desirable. It is necessary because SSH channels and subsystems can only be opened on the SSH server. It is desirable because it conveniently leverages infrastructure that may be deployed for host-key verification and user authentication.

Call home is useful for both initial deployment and on-going device management and may be used to enable any of the following scenarios:

- The network element may proactively call home after being powered on for the first time to register itself with its management system.

- The network element may access the network in a way that dynamically assigns it an IP address and it doesn’t register its assigned IP addressed to a mapping service.

- The network element may be configured in "stealth mode" and thus doesn’t have any open ports for the management system to connect to.

- The network element may be deployed behind a firewall that doesn’t allow SSH access to the internal network.

- The network element may be deployed behind a firewall that implements network address translation (NAT) for all internal network IP addresses, thus complicating the ability for a management system to connect to it.

- The operator may prefer to have network elements initiate management connections believing it is easier to secure one open-port in the data center than to have an open port on each network element in the network.

One key benefit of using SSH as the transport protocol is its ability to multiplex an unspecified number of independently flow-controlled TCP sessions [RFC4254]. This is valuable as the network element only needs to be configured to initiate a single call home connection to a management system, regardless the number of NETCONF channels the management system wants to open.

4. Protocol

The NETCONF server's perspective (e.g., the network element)

- The NETCONF server initiates a TCP connection to the NETCONF client on the IANA-assigned SSH for NETCONF Call Home port YYYY.

- The TCP connection is accepted and a TCP session is established.

- Using this TCP connection, the NETCONF server immediately starts the SSH server protocol. That is, the next message sent on the TCP stream is SSH's Protocol Version Exchange message (section 4.2, [RFC4253]).

- The SSH connection is established.

The NETCONF client's perspective (e.g., the management system)

- The NETCONF client listens for TCP connections on the IANA-assigned NETCONF over SSH Call Home port YYYY.

- The NETCONF client accepts an incoming TCP connection and a TCP session is established.

- Using this TCP connection, the NETCONF client immediately starts the SSH Client protocol, starting with sending the SSH's Protocol Version Exchange message (section 4.2, [RFC4253]).

- The SSH connection is established.

5. SSH Server Identification and Verification

When the management system accepts a new incoming TCP connection on the NETCONF over SSH Call Home port, it starts the SSH client protocol. As the SSH client, it MUST authenticate the SSH server, by both identifying the network element and verifying its SSH host key.

Due to call home having the network element initiate the TCP connection, the management system MAY identify the remote peer using the source IP address of the TCP connection. However, identifying the remote peer using the source IP address of the TCP connection is NOT RECOMMENDED as it can only work in networks that use known static addresses.

To support network elements having dynamically-assigned IP addresses, or deployed behind gateways that translate their IP addresses (e.g., NAT), the management system MAY identify the device using its SSH host key. For instance, a fingerprint of the network element's host key could itself be used as an identifier since each device has a statistically unique host key. However, identifying the remote peer using its host key directly is NOT RECOMMENDED as it requires the host key to be manually verified the first time the network element connects and anytime its host key changes thereafter.

Yet another option for identifying the network element is for its host key to encode the network element's identity, such as if the host key were a certificate. This option enables the host key to change over time, so long as it continues to encode the same identity, but brings the next issue of how the management system can verify the network element's host key is authentic.

The security of SSH is anchored in the ability for the SSH client to verify the SSH server's host key. Typically this is done by comparing the host key presented by the SSH server with one that was previously configured on the SSH client, looking it up in a local database using the identity of the SSH client as the lookup key. Nothing changes regarding this requirement due to the direction reversal of the underlying TCP connection. To ensure security, the management system MUST verify the network element's SSH host key each time a SSH session is established.

However, configuring distinct host keys on the management system doesn't scale well, which is an important consideration to a network management system. A more scalable strategy for the management system is for the network element's manufacturer to sign the network-element's host key with a common trusted key, such as a certificate authority. Then, when the network-element is deployed, the management system only needs to trust a single certificate, which vouches for the authenticity of the various network element host keys.

Since both the identification and verification issues are addressed using certificates, this draft RECOMMENDS network elements use a host key that can encode a unique identifier (e.g., its serial number) and be signed by a common trust anchor (e.g., a certificate authority). Examples of suitable public host keys are the X.509v3 keys defined in defined in [RFC6187] and the PGP keys defined in [RFC4253].

6. Device Configuration

How to configure a device to initiate a NETCONF over SSH Call Home connection is outside the scope of this document, as implementations can support this protocol using a proprietary configuration data model. That said, a YANG [RFC6020] model to configure NETCONF over SSH Call Home is specified in [draft-ietf-netconf-server-model].

7. Security Considerations

This RFC deviates from standard SSH protocol usage by allowing the SSH server to initiate the TCP connection. This conflicts with section 4 of the SSH Transport Layer Protocol RFC [RFC4253], which states "The client initiates the connection". However this statement is made without rationalization and it's not clear how it impacts the security of the protocol, so this section analyzes the security offered by having the client initiate the connection.

First, assuming the SSH server is not using a public host key algorithm that certifies its identity, the security of the protocol doesn't seem to be sensitive to which peer initiates the connection. That is, it is still the case that reliable distribution of host keys (or their fingerprints) should occur prior to first connection and that verification for subsequent connections happens by comparing the host keys in a locally cached database. It does not seem to matter if the SSH server's host name is derived from user-input or extracted from the TCP layer, potentially via a reverse-DNS lookup. Once the host name-to-key association is stored in a local database, no man-in-the-middle attack is possible due to the attacker being unable to guess the real SSH server's private key (Section 9.3.4 (Man-in-the-middle) of [RFC4251]).

That said, this RFC recommends implementations use a public host key algorithm that certifies the SSH server's identity. The identity can be any unique identifier, such as a device's serial number or a deployment-specific value. If this recommendation is followed, then no information from the TCP layer would be needed to lookup the device in a local database and therefore the directionality of the TCP layer is clearly inconsequential.

The SSH protocol negotiates which algorithms it will use during key exchange (Section 7.1 (Algorithm Negotiation) in [RFC4253]). The algorithm selected is essentially the first compatible algorithm listed by the SSH client that is also listed by the SSH server. For a network management application, there may be a need to advertise a large number of algorithms to be compatible with the various devices it manages. The SSH client SHOULD order its list of public host key algorithms such that all the certifiable public host key algorithms are listed first. Additionally, when possible, SSH servers SHOULD only list certifiable public host key algorithms. Note that since the SSH server would have to be configured to know which IP address it is to connect to, it is expected that it will also be configured to know which host key algorithm to use for the particular application, and hence only needs to list just that one public host key algorithm.

This RFC suggests implementations can use a device's serial number as a form of identity. A potential concern with using a serial number is that the SSH protocol passes the SSH server's host-key in the clear and many times serial numbers encode revealing information about the device, such as what kind of device it is and when it was manufactured. While there is little security in trying to hide this information from an attacker, it is understood that some deployments may want to keep this information private. If this is a concern, deployments SHOULD use an alternate unique identifier, if even just the hash of the device's serial number.

An attacker could DoS the application by having it perform computationally expensive operations, before deducing that the attacker doesn't posses a valid key. This is no different than any secured service and all common precautions apply (e.g., blacklisting the source address after a set number of unsuccessful login attempts).

8. IANA Considerations

This document requests that IANA assigns a TCP port number in the "Registered Port Numbers" range with the service name "netconf-ssh-ch". This port will be the default port for NETCONF over SSH Call Home protocol and will be used when the NETCONF server is to initiate a connection to a NETCONF client using SSH. Below is the registration template following the rules in [RFC6335].

```
Service Name:           netconf-ssh-ch
Transport Protocol(s):  TCP
Assignee:               IESG <iesg@ietf.org>
Contact:                IETF Chair <chair@ietf.org>
Description:            NETCONF over SSH Call Home
Reference:              RFC XXXX
Port Number:            YYYY
```

9. Acknowledgements

The author would like to thank for following for lively discussions on list and in the halls (ordered by last name): Andy Bierman, Martin Bjorklund, Mehmet Ersue, Wes Hardaker, Stephen Hanna, David Harrington, Jeffrey Hutzelman, Radek Krejci, Alan Luchuk, Mouse, Russ Mundy, Tom Petch, Peter Saint-Andre, Joe Touch, Sean Turner, Bert Wijnen.

10. References

10.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels ", BCP 14, RFC 2119, March 1997.

[RFC4250] Lehtinen, S. and C. Lonvick, "The Secure Shell (SSH) Protocol Assigned Numbers ", RFC 4250, December 2005.

[RFC4251] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture ", RFC 4251, January 2006.

[RFC4252] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Authentication Protocol ", RFC 4252, January 2006.

[RFC4253] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Transport Layer Protocol ", RFC 4253, January 2006.

[RFC4254] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Connection Protocol ", RFC 4254, January 2006.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF) ", RFC 6020, October 2010.

[RFC6187] Igoe, K. and D. Stebila, "X.509v3 Certificates for Secure Shell Authentication ", RFC 6187, March 2011.

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "NETCONF Configuration Protocol", RFC 6241, June 2011.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", RFC 6335, August 2011.

10.2. Informative References

[draft-ietf-netconf-server-model]
Watsen, K. and J. Schoenwaelder, "A YANG Data Model for NETCONF Server Configuration", RFC 6242, June 2011.

Appendix A. Change Log

A.1. 05 to 06

Changed title to "NETCONF Call Home using SSH"

Revised the Abstract and Introduction to better explain what the document regards.

Changed "MUST" to "SHOULD" in the Applicability Statement.

Added a "Draft Naming" section explaining why, despite its name, reversing SSH is nowhere in the text

Added PGP keys as another kind of SSH host key encoding identity and signed by a trust anchor.

Revised the Device Considerations section to more clearly explain why a device configuration data model is out of scope, and hence an Informative reference.

Clarified Security Considerations section on use of serial numbers.

A.2. 04 to 05

Changed "Reverse SSH" to "Call Home"

Added references to Applicability Statement

A.3. 03 to 04

Changed title to "Reverse SSH for NETCONF Call Home" (changed again in -05)

Removed statement on how other SSH channels might be used for other protocols

Improved language on how the management system, as the SSH client, MUST authenticate the SSH server

Clarified that identifying the network element using source IP address is NOT RECOMMENDED

Clarified that identifying the NE using simple certificate comparison is NOT RECOMMENDED

Device Configuration section now more clearly states that the YANG model is out of scope

Change requested port name to "netconf-ssh-ch"

General edits for grammer, capitalization, and spellings

A.4. 02 to 03

Updated Device Configuration section to reference [draft-ietf-netconf-server-model]

A.5. 01 to 02

Added Applicability Statement

Removed references to ZeroConf / ZeroTouch

Clarified the protocol section

Added a section for identification and verification

A.6. 00 to 01

Removed the hmac-* family of algorithms

Author's Address

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

NETCONF Working Group
Internet-Draft
Obsoletes: 5539 (if approved)
Intended status: Standards Track
Expires: October 12, 2015

M. Badra
Zayed University
A. Luchuk
SNMP Research, Inc.
J. Schoenwaelder
Jacobs University Bremen

April 10, 2015

Using the NETCONF Protocol over Transport Layer Security (TLS) with Mutual X.509 Authentication

draft-ietf-netconf-rfc5539bis-10

Abstract

The Network Configuration Protocol (NETCONF) provides mechanisms to install, manipulate, and delete the configuration of network devices. This document describes how to use the Transport Layer Security (TLS) protocol with mutual X.509 authentication to secure the exchange of NETCONF messages. This revision of RFC 5539 documents the new message framing used by NETCONF 1.1 and it obsoletes RFC 5539.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The NETCONF protocol [RFC6241] defines a mechanism through which a network device can be managed. NETCONF is connection-oriented, requiring a persistent connection between peers. This connection must provide integrity, confidentiality, peer authentication, and reliable, sequenced data delivery.

This document defines how NETCONF messages can be exchanged over Transport Layer Security (TLS) [RFC5246]. Implementations MUST support mutual TLS certificate-based authentication [RFC5246]. This assures the NETCONF server of the identity of the principal who wishes to manipulate the management information. It also assures the NETCONF client of the identity of the server for which it wishes to manipulate the management information.

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 [RFC2119].

2. Connection Initiation

The peer acting as the NETCONF client MUST act as the TLS client. The TLS client actively opens the TLS connection and the TLS server passively listens for the incoming TLS connections. The well-known TCP port number 6513 is used by NETCONF servers to listen for TCP connections established by NETCONF over TLS clients. The TLS client MUST send the TLS ClientHello message to begin the TLS handshake. The TLS server MUST send a CertificateRequest in order to request a certificate from the TLS client. Once the TLS handshake has finished, the client and the server MAY begin to exchange NETCONF messages. Client and server identity verification is done before the NETCONF <hello> message is sent. This means that the identity verification is completed before the NETCONF session is started.

3. Message Framing

All NETCONF messages MUST be sent as TLS "application data". It is possible that multiple NETCONF messages be contained in one TLS record, or that a NETCONF message be transferred in multiple TLS records.

The previous version of this document [RFC5539] used the framing sequence defined in [RFC4742]. This version aligns with [RFC6242] and adopts the framing protocol defined in [RFC6242] as follows:

The NETCONF <hello> message MUST be followed by the character sequence]]>]]>. Upon reception of the <hello> message, the peers inspect the announced capabilities. If the :base:1.1 capability is advertised by both peers, the chunked framing mechanism defined in Section 4.2 of [RFC6242] is used for the remainder of the NETCONF session. Otherwise, the old end-of-message-based mechanism (see Section 4.3 of [RFC6242]) is used.

4. Connection Closure

A NETCONF server will process NETCONF messages from the NETCONF client in the order in which they are received. A NETCONF session is closed using the <close-session> operation. When the NETCONF server processes a <close-session> operation, the NETCONF server SHALL respond and close the TLS session as described in Section 7.2.1 of [RFC5246].

5. Certificate Validation

Both peers MUST use X.509 certificate path validation [RFC5280] to verify the integrity of the certificate presented by the peer. The presented X.509 certificate may also be considered valid if it matches one obtained by another trusted mechanism, such as using a locally configured certificate fingerprint. If X.509 certificate path validation fails and the presented X.509 certificate does not match a certificate obtained by a trusted mechanism, the connection MUST be terminated as defined in [RFC5246].

6. Server Identity

The NETCONF client MUST check the identity of the server according to Section 6 of [RFC6125].

7. Client Identity

The NETCONF server MUST verify the identity of the NETCONF client to ensure that the incoming request to establish a NETCONF session is legitimate before the NETCONF session is started.

The NETCONF protocol [RFC6241] requires that the transport protocol's authentication process results in an authenticated NETCONF client identity whose permissions are known to the server. The authenticated identity of a client is commonly referred to as the NETCONF username. The following algorithm is used by the NETCONF server to derive a NETCONF username from a certificate. (Note that the algorithm below is the same as the one described in the SNMP-TLS-TM-MIB MIB module defined in [RFC6353] and in the ietf-x509-cert-to-name YANG module defined in [RFC7407].)

(a) The server maintains an ordered list of mappings of certificates to NETCONF usernames. Each list entry contains

  * a certificate fingerprint (used for matching the presented certificate),

  * a map type (indicates how the NETCONF username is derived from the certificate), and

  * optional auxiliary data (used to carry a NETCONF username if the map type indicates the user name is explicitly configured).

(b) The NETCONF username is derived by considering each list entry in order. The fingerprint member of the current list entry determines whether the current list entry is a match:

  1. If the list entry's fingerprint value matches the fingerprint of the presented certificate, then consider the list entry as a successful match.

2. If the list entry's fingerprint value matches that of a locally held copy of a trusted CA certificate, and that CA certificate was part of the CA certificate chain to the presented certificate, then consider the list entry as a successful match.

(c) Once a matching list entry has been found, the map type of the current list entry is used to determine how the username associated with the certificate should be determined. Possible mapping options are:

A. The username is taken from the auxiliary data of the current list entry. This means the username is explicitely configured (map type 'specified').

B. The subjectAltName's rfc822Name field is mapped to the username (map type 'san-rfc822-name'). The local part of the rfc822Name is used unaltered but the host-part of the name must be converted to lowercase.

C. The subjectAltName's dNSName is mapped to the username (map type 'san-dns-name'). The characters of the dNSName are converted to lowercase.

D. The subjectAltName's iPAddress is mapped to the username (map type 'san-ip-address'). IPv4 addresses are converted into decimal-dotted quad notation (e.g., '192.0.2.1'). IPv6 addresses are converted into a 32-character all lowercase hexadecimal string without any colon separators.

E. Any of the subjectAltName's rfc822Name, dNSName, iPAddress is mapped to the username (map type 'san-any'). The first matching subjectAltName value found in the certificate of the above types MUST be used when deriving the name.

F. The certificate's CommonName is mapped to the username (map type 'common-name'). The CommonName is converted to UTF-8 encoding. The usage of CommonNames is deprecated and users are encouraged to use subjectAltName mapping methods instead.

(d) If it is impossible to determine a username from the list entry's data combined with the data presented in the certificate, then additional list entries MUST be searched looking for another potential match. Similarily, if the username does not comply to the NETCONF requirements on usernames [RFC6241], then additional list entries MUST be searched looking for another potential match. If there are no further list entries, the TLS session MUST be terminated.

The username provided by the NETCONF over TLS implementation will be made available to the NETCONF message layer as the NETCONF username without modification.

The NETCONF server configuration data model [I-D.ietf-netconf-server-model] covers NETCONF over TLS and provides further details such as certificate fingerprint formats exposed to network configuration systems.

8. Cipher Suites

Implementations MUST support TLS 1.2 [RFC5246] and are REQUIRED to support the mandatory-to-implement cipher suite. Implementations MAY implement additional TLS cipher suites that provide mutual authentication [RFC5246] and confidentiality as required by NETCONF [RFC6241]. Implementations SHOULD follow the recommendations given in [I-D.ietf-uta-tls-bcp].

9. Security Considerations

NETCONF is used to access configuration and state information and to modify configuration information, so the ability to access this protocol should be limited to users and systems that are authorized to view the NETCONF server’s configuration and state or to modify the NETCONF server’s configuration.

Configuration or state data may include sensitive information, such as usernames or security keys. So, NETCONF requires communications channels that provide strong encryption for data privacy. This document defines a NETCONF over TLS mapping that provides for support of strong encryption and authentication. The security considerations for TLS [RFC5246] and NETCONF [RFC6241] apply here as well.

NETCONF over TLS requires mutual authentication. Neither side should establish a NETCONF over TLS connection with an unknown, unexpected, or incorrect identity on the opposite side. Note that the decision whether a certificate presented by the client is accepted can depend on whether a trusted CA certificate is white listed (see Section 7). If deployments make use of this option, it is recommended that the white listed CA certificate is used only to issue certificates that are used for accessing NETCONF servers. Should the CA certificate be used to issue certificates for other purposes, then all certificates created for other purposes will be accepted by a NETCONF server as well, which is likely not suitable.

This document does not support third-party authentication (e.g., backend Authentication, Authorization, and Accounting (AAA) servers) due to the fact that TLS does not specify this way of authentication and that NETCONF depends on the transport protocol for the authentication service. If third-party authentication is needed, the SSH transport [RFC6242] can be used.

RFC 5539 assumes that the end-of-message (EOM) sequence,]]>]]>, cannot appear in any well-formed XML document, which turned out to be mistaken. The EOM sequence can cause operational problems and open space for attacks if sent deliberately in NETCONF messages. It is however believed that the associated threat is not very high. This document still uses the EOM sequence for the initial <hello> message to avoid incompatibility with existing implementations. When both peers implement :base:1.1 capability, a proper framing protocol (chunked framing mechanism; see Section 3) is used for the rest of the NETCONF session, to avoid injection attacks.

10. IANA Considerations

Based on the previous version of this document, RFC 5539, IANA has assigned a TCP port number (6513) in the "Registered Port Numbers" range with the service name "netconf-tls". This port will be the default port for NETCONF over TLS, as defined in Section 2. Below is the registration template following the rules in [RFC6335].

```
Service Name:            netconf-tls
Transport Protocol(s):   TCP
Assignee:                IESG <iesg@ietf.org>
Contact:                 IETF Chair <chair@ietf.org>
Description:             NETCONF over TLS
Reference:               RFC XXXX
Port Number:             6513
```

[[CREF1: RFC Editor: Please replace XXXX above with the allocated RFC number and remove this comment. --JS]]

11. Acknowledgements

The authors like to acknowledge Martin Bjorklund, Olivier Coupelon, Mehmet Ersue, Stephen Farrell, Miao Fuyou, Ibrahim Hajjeh, David Harrington, Sam Hartman, Alfred Hoenes, Simon Josefsson, Barry Leiba, Tom Petch, Eric Rescorla, Dan Romascanu, Kent Watsen, Bert Wijnen, Stefan Winter and the NETCONF mailing list members for their comments on this document. Charlie Kaufman, Pasi Eronen, and Tim Polk provided a thorough review of previous versions of this document.

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

12. References

12.1. Normative References

[I-D.ietf-uta-tls-bcp]
Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of TLS and DTLS", draft-ietf-uta-tls-bcp-09 (work in progress), February 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[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, May 2008.

[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, March 2011.

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, August 2011.

12.2. Informative References

[I-D.ietf-netconf-server-model]
Watsen, K. and J. Schoenwaelder, "NETCONF Server and RESTCONF Server Configuration Models", draft-ietf-netconf-server-model-06 (work in progress), February 2015.

[RFC4742] Wasserman, M. and T. Goddard, "Using the NETCONF Configuration Protocol over Secure SHell (SSH)", RFC 4742, December 2006.

[RFC5539] Badra, M., "NETCONF over Transport Layer Security (TLS)", RFC 5539, May 2009.

[RFC6353] Hardaker, W., "Transport Layer Security (TLS) Transport Model for the Simple Network Management Protocol (SNMP)", STD 78, RFC 6353, July 2011.

[RFC7407] Bjorklund, M. and J. Schoenwaelder, "A YANG Data Model for SNMP Configuration", RFC 7407, December 2014.

Appendix A. Changes from RFC 5539

This section summarizes major changes between this document and RFC 5539.

- o Documented that NETCONF over TLS uses the new message framing if both peers support the :base:1.1 capability.
- o Removed redundant text that can be found in the TLS and NETCONF specifications and restructured the text. Alignment with [RFC6125].
- o Added a high-level description how NETCONF usernames are derived from certificates.
- o Removed the reference to BEEP.

Authors' Addresses

Mohamad Badra
Zayed University

Email: mbadra@gmail.com

Alan Luchuk
SNMP Research, Inc.
3001 Kimberlin Heights Road
Knoxville, TN 37920
USA

Phone: +1 865 573 1434
Email: luchuk@snmp.com
URI: http://www.snmp.com/

Juergen Schoenwaelder
Jacobs University Bremen
Campus Ring 1
28759 Bremen
Germany

Phone: +49 421 200 3587
Email: j.schoenwaelder@jacobs-university.de
URI: http://www.jacobs-university.de/

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 18, 2014

K. Watsen
Juniper Networks
J. Schoenwaelder
Jacobs University Bremen

February 14, 2014

A YANG Data Model for NETCONF Server Configuration
draft-kwatsen-netconf-server-01

Abstract

This draft defines a NETCONF server configuration data model. This data model enables configuration of the NETCONF service itself, including which transports it supports, what ports they listen on, whether they support device-initiated connections, and associated parameters.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This draft defines a NETCONF [RFC6241] server configuration data model. This data model enables configuration of the NETCONF service itself, including which transports it supports, what ports they listen on, whether they support device-initiated connections, and associated parameters.

1.1. Terminology

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

1.2. Tree Diagrams

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

- o Brackets "[" and "]" enclose list keys.
- o Abbreviations before data node names: "rw" means configuration (read-write) and "ro" state data (read-only).
- o 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 (":").

2. Objectives

The primary purpose of this YANG module is to enable the configuration of the NETCONF service on the device. This scope includes both transport-independent and transport-specific configuration parameters.

2.1. Support all NETCONF Transports

The YANG module should support all current NETCONF transports, namely NETCONF over SSH [RFC6242] and NETCONF over TLS [I-D.ietf-netconf-rfc5539bis], and be extensible to support future transports as necessary.

Since implementations may not support all transports, the module should use YANG "feature" statements so that each implementation can advertise which transports it actually supports.

2.2. Align Transport-Specific Configurations

While each transport is unique in its protocol and may have some distinct configurations, there remains a significant overlap between them. Thus the YANG module should use "grouping" statements so that the common aspects can be configured similarly.

2.3. Support Transport-Independent Configuration

Since some NETCONF server configurations may be independent of any transport, the module should define a location for these transport-independent values to be configured.

2.4. Support both Inbound and Outbound Connections

Historically, NETCONF only supported the device opening a port to listen for inbound client connections. However, the NETCONF working group is actively defining support for devices to initiate outbound connections (e.g., "call home"). Thus, the module should enable the configuration of both inbound and outbound connections.

Since implementations may not support both inbound and outbound connections, the module should use YANG "feature" statements so that each implementation can advertise the type of connections it actually supports.

2.5. For Device-Initiated Outbound Connections

The following objectives only pertain to support for device-initiated outbound connections.

2.5.1. Support More than One Application

A device may be managed by more than one northbound applications. For instance, a deployment may have one application for provisioning and another for fault monitoring. Therefore, when it is desired for a device to initiate management connections, it should be able to do so for more than one application.

2.5.2. Support Applications Having More than One Server

An application managing a device may implement a high-availability strategy employing a multiplicity of active and/or passive servers. Therefore, when it is desired for a device to initiate connections to the application, it should be able to connect to any of the applications servers.

2.5.3. Support a Reconnection Strategy

Assuming an application has more than one server, then it becomes necessary to understand how a device should reconnect to the application should it lose its connection to one of the application's servers. Of primary interest is if the device should start with first server defined in a user-ordered list of servers or with the last server it was connected to. Secondary settings might specify the frequency of attempts and number of attempts per server. Therefore, a reconnection strategy should be configurable.

Note that 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 the device how to get connected to another server.

2.5.4. Support both Persistent and Periodic Connections

Applications may vary greatly on how frequently they need to interact with a device, how responsive interactions with devices need to be, and how many simultaneous connections they can support. Some applications may need a persistent connection to devices to optimize real-time interactions, while others are satisfied with periodic interactions and reduced resources required. Therefore, when it is necessary for devices to initiate connections, the type of connection desired should be configured.

2.5.5. Keep-Alives for Persistent Connections

If a persistent connection is desired, it is the responsibility of the connection-initiator to actively test the connection for aliveness. However, there is a balance between the frequency of the tests and the networking overhead they generate. The appropriate balance can only be determined by the application, based on its interaction requirements. Therefore, for persistent connections, keep-alive settings should be configurable on a per-application basis.

2.5.6. Customizations for Periodic Connections

If a periodic connection is desired, it is necessary for the device to know how often it should connect. This delay essentially determines how long the application might have to wait to send data to the device. Note, this setting does not constrain how often the device must wait to send data to the application, as the device should immediately connect to the application whenever it has data to send to it.

A common communication pattern is that one data transmission is many times closely followed by another. For instance, if the device needs to send a notification message, there’s a high probability that it will send another shortly thereafter. Likewise, the application may have a sequence of pending messages to send. Thus, it should be possible for a device to hold a connection open until some amount of time of no data being transmitted as transpired.

3. Data Model Overview

3.1. The "listen" Grouping

To enable transports to configure listening on one or more ports in a common way, this grouping is defined. Being a grouping enables each transport-specific data-model to augment it as needed (e.g., to specify a default for the "port" values), as well as enable implementations to advertise support for listening for inbound connections using a YANG feature.

```
+--rw listen
   +--rw (one-or-many)?
      +--:(one-port)
      |  +--rw port?         inet:port-number
      +--:(many-ports)
         +--rw interface* [address]
            +--rw address    inet:ip-address
            +--rw port?      inet:port-number
```

3.2. The "call-home" Grouping

To enable transports to configure initiating connections to remote applications in a common way, this grouping is defined. Being a grouping enables each transport-specific data-model to augment it as needed (e.g., to specify a default port value, lists of algorithms to advertise, etc.), as well as enable implementations to advertise support for listening for inbound connections using a YANG feature.

```
+--rw call-home
   +--rw applications
      +--rw application* [name]
         +--rw name                  string
         +--rw description?          string
         +--rw servers
         |  +--rw server* [address]
```

```
            |     +--rw address    inet:host
            |     +--rw port?      inet:port-number
            +--rw connection-type
            |  +--rw (connection-type)?
            |     +--:(persistent-connection)
            |     |  +--rw persistent
            |     |     +--rw keep-alives
            |     |        +--rw interval-secs?   uint8
            |     |        +--rw count-max?       uint8
            |     +--:(periodic-connection)
            |        +--rw periodic
            |           +--rw timeout-mins?   uint8
            |           +--rw linger-secs?    uint8
            +--rw reconnect-strategy
               +--rw start-with?      enumeration
               +--rw interval-secs?   uint8
               +--rw count-max?       uint8
```

3.3. The SSH Subtree

The SSH subtree uses both the "listen" and "call-home" groupings mentioned above. Support for the SSH transport is advertised by the "ssh" feature, while listening for clients and calling home are advertised by the "inbound-ssh" and "outbound-ssh" features respectively. The SSH subtree augments the "call-home" grouping by adding a "host-keys" container. Also, though not visible in the tree output below, this subtree refines all the port values with a suitable default (i.e., 830).

```
   +--rw netconf
      +--rw ssh {ssh}?
         +--rw listen {inbound-ssh}?
         |  +--rw (one-or-many)?
         |     +--:(one-port)
         |     |  +--rw port?        inet:port-number
         |     +--:(many-ports)
         |        +--rw interface* [address]
         |           +--rw address    inet:ip-address
         |           +--rw port?      inet:port-number
         +--rw call-home {outbound-ssh}?
            +--rw applications
               +--rw application* [name]
                  +--rw name                  string
                  +--rw description?          string
                  +--rw servers
                  |  +--rw server* [address]
                  |     +--rw address    inet:host
                  |     +--rw port?      inet:port-number
```

```
                  +--rw connection-type
                  |  +--rw (connection-type)?
                  |     +--:(persistent-connection)
                  |     |  +--rw persistent
                  |     |     +--rw keep-alives
                  |     |        +--rw interval-secs?   uint8
                  |     |        +--rw count-max?       uint8
                  |     +--:(periodic-connection)
                  |        +--rw periodic
                  |           +--rw timeout-mins?   uint8
                  |           +--rw linger-secs?    uint8
                  +--rw reconnect-strategy
                  |  +--rw start-with?      enumeration
                  |  +--rw interval-secs?   uint8
                  |  +--rw count-max?       uint8
                  +--rw host-keys
                     +--rw host-key* [name]
                        +--rw name    string
```

3.4. The TLS Subtree

The TLS subtree uses both the "listen" and "call-home" groupings mentioned above, while also defining containers for certificate and pre-shared key mappings. Support for the TLS transport is advertised by the "tls" feature, while listening for clients and calling home are advertised by the "inbound-tls" and "outbound-tls" features respectively. Also, though not visible in the tree output below, this submodule refines all the port values with a suitable defaults (e.g., 6513).

```
+--rw netconf
   +--rw tls {tls}?
      +--rw listen {inbound-tls}?
      |  +--rw (one-or-many)?
      |     +--:(one-port)
      |     |  +--rw port?        inet:port-number
      |     +--:(many-ports)
      |        +--rw interface* [address]
      |           +--rw address    inet:ip-address
      |           +--rw port?      inet:port-number
      +--rw call-home {outbound-tls}?
      |  +--rw applications
      |     +--rw application* [name]
      |        +--rw name                  string
      |        +--rw description?          string
      |        +--rw servers
      |        |  +--rw server* [address]
      |        |     +--rw address    inet:host
```

```
         |        |     +--rw port?      inet:port-number
         |        +--rw connection-type
         |        |  +--rw (connection-type)?
         |        |     +--:(persistent-connection)
         |        |     |  +--rw persistent
         |        |     |     +--rw keep-alives
         |        |     |        +--rw interval-secs?   uint8
         |        |     |        +--rw count-max?       uint8
         |        |     +--:(periodic-connection)
         |        |        +--rw periodic
         |        |           +--rw timeout-mins?   uint8
         |        |           +--rw linger-secs?    uint8
         |        +--rw reconnect-strategy
         |           +--rw start-with?      enumeration
         |           +--rw interval-secs?   uint8
         |           +--rw count-max?       uint8
         +--rw cert-maps {tls-map-certificates}?
         |  +--rw cert-to-name* [id]
         |     +--rw id             uint32
         |     +--rw fingerprint    x509c2n:tls-fingerprint
         |     +--rw map-type       identityref
         |     +--rw name           string
         +--rw psk-maps {tls-map-pre-shared-keys}?
            +--rw psk-map* [psk-identity]
               +--rw psk-identity        string
               +--rw user-name           nacm:user-name-type
               +--rw not-valid-before?   yang:date-and-time
               +--rw not-valid-after?    yang:date-and-time
               +--rw key                 yang:hex-string
```

4. NETCONF Server YANG Module

This YANG module imports YANG extensions from [RFC6536], and imports YANG types from [RFC6991] and a YANG grouping from [I-D.ietf-netmod-snmp-cfg].

```
module ietf-netconf-server {

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

  import ietf-yang-types {
    prefix yang;                // RFC 6991
  }
  import ietf-inet-types {
    prefix inet;                // RFC 6991
  }
  import ietf-x509-cert-to-name {
```

```
     prefix x509c2n;             // I-D.ietf-netconf-rfc5539bis
   }
   import ietf-netconf-acm {
     prefix nacm;                // RFC 6536
   }

   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: Bert Wijnen
               <mailto:bertietf@bwijnen.net>

     Editor:   Juergen Schoenwaelder
               <mailto:j.schoenwaelder@jacobs-university.de>

               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.";
   // RFC Ed.: replace XXXX with actual RFC number and
   // remove this note

   // RFC Ed.: please update the date to the date of publication

   revision "2014-01-24" {
     description
```

```
        "Initial version";
      reference
        "RFC XXXX: A YANG Data Model for NETCONF Server Configuration";
    }

    /*
     * Features
     */

    feature ssh {
      description
        "A server implements this feature if it supports NETCONF
         over Secure Shell (SSH).";
      reference
        "RFC 6242: Using the NETCONF Protocol over Secure Shell (SSH)";
    }

    feature inbound-ssh {
      description
        "The inbound-ssh feature indicates that the server can
         open a port to listen for incoming client connections.";
    }

    feature outbound-ssh {
      description
        "The outbound-ssh feature indicates that the server can
         connect to a client.";
      reference
        "RFC XXXX: Reverse Secure Shell (Reverse SSH)";
    }

    feature tls {
      description
        "A server implements this feature if it supports NETCONF
         over Transport Layer Security (TLS).";
      reference
        "RFC XXXX: NETCONF over Transport Layer Security (TLS)";
    }

    feature inbound-tls {
      description
        "The inbound-tls feature indicates that the server can
         open a port to listen for incoming client connections.";
    }

    feature outbound-tls {
      description
        "The outbound-tls feature indicates that the server can
```

```
         connect to a client.";
     }

     feature tls-map-certificates {
       description
        "The tls-map-certificates feature indicates that the
         server implements mapping X.509 certificates to NETCONF
         usernames.";
     }

     feature tls-map-pre-shared-keys {
       description
        "The tls-map-pre-shared-keys feature indicates that the
         server implements mapping TLS pre-shared keys to NETCONF
         usernames.";
     }

     /*
      * Groupings
      */

      grouping listen-config {
        description
         "Provides a choice of configuring one of more ports
          to listen for incoming client connections.";

        choice one-or-many {
          default one-port;
          case one-port {
            leaf port {
              type inet:port-number;
              description
               "The port number the server listens on on all
                interfaces.";
            }
          }

          case many-ports {
            list interface {
              key "address";
              leaf address {
                type inet:ip-address;
                  mandatory true;
                  description
                   "The local IP address of the interface to listen
                    on.";
                }
                leaf port {
```

```
                  type inet:port-number;
                  description
                   "The local port number on this interface the
                    server listens on.";
                }
              }
            }
          }
        }

        grouping call-home-config {

          container applications {
            description
             "A list of applications the device iniates connections
              to. The configuration for each application specifies
              its details, including its servers, the type of
              connection to maintain, and the reconnection strategy
              to use.";

            list application {
              key name;
   //           min-elements 1;  // this forces <call-home>?!
              leaf name {
                type string {
                  length 1..64;  // XXX why these limits?
                }
                mandatory true;
                description
                 "An arbitrary name for the application the device
                  is connecting to.";
              }
              leaf description {
                type string;
                description
                  "An optional description for the application.";
              }
              container servers {
                description
                 "An ordered listing of the application’s servers
                  that the device should attempt connecting to.";
                list server {
                  key address;
                  min-elements 1;
                  ordered-by user;
                  leaf address {
                    type inet:host;
                    mandatory true;
```

```
                  description
                   "The address or domain-name of the server.";
                }
                leaf port {
                  type inet:port-number;
                  description
                   "The IP port for this server. The device will use
                    the IANA-assigned well-known port if not specified.";
                }
              }
            }
            container connection-type {
              description
               "Indicates the application’s preference for how the
                device’s connection is maintained.";
              choice connection-type {
                default persistent-connection;

                case persistent-connection {
                  container persistent {
                    description
                     "Maintain a persistent connection to the
                      application. If the connection goes down,
                      immediately start trying to reconnect to it,
                      using the reconnection strategy.

                      This connection type minimizes any
                      application-to-server data-transfer delay,
                      albeit at the expense of holding resources
                      longer.";
                    container keep-alives {
                      leaf interval-secs {
                        type uint8;
                        units seconds;
                        default 15;
                        description
                         "Sets a timeout interval in seconds after which
                          if no data has been received from the
                          application, a message will be sent to request
                          a response from the application.  A value of
                          ’0’ indicates that no keep-alive messages
                          should be sent.";
                      }
                      leaf count-max {
                        type uint8;
                        default 3;
                        description
                         "Sets the number of keep-alive messages that may
```

```
                        be sent without receiving any data from the
                        application before assuming the application is
                        no longer alive.  If this threshold is reached,
                        the transport-level connection will be
                        disconnected (thus triggering the reconnection
                        strategy).  The interval timer is reset after
                        each transmission, thus an unresponsive
                        application will be disconnected after
                        approximately count-max * interval-secs
                        seconds.";
                    }
                  }
                }
              }

              case periodic-connection {
                container periodic {
                  description
                   "Periodically connect to application, using the
                    reconnection strategy, so it can flush any pending
                    data it may be holding. This connection type
                    minimizes resources held open, albeit at the
                    expense of longer application-to-server
                    data-transfer delay.  Note that for
                    server-to-application data, the data should be
                    sent immediately, connecting to application first
                    if not already.";
                  leaf timeout-mins {
                    type uint8;
                    units minutes;
                    default 5;
                    description
                     "The maximum amount of unconnected time the
                      device will wait until establishing a
                      connection to the application again. The
                      device may establish a connection before this
                      time if it has data it needs to send to the
                      application. Note: this value differs from
                      the reconnection strategy’s interval-secs
                      value.";
                  }
                  leaf linger-secs {
                    type uint8;
                    units seconds;
                    default 30;
                    description
                     "The amount of time the device should wait after
                      last receiving data from or sending data to the
```

```
                  application before closing its connection to it.
                  This is an optimization to prevent unnecessary
                  connections.";
              }
            }
          }
        }
      }

      // XXX
      // Should we have something smarter as the reconnect
      // strategy, e.g. an exponential backoff?

      container reconnect-strategy {
        description
         "The reconnection strategy guides how a device reconnects
          to an application, after losing a connection to it, even
          if due to a reboot.  The device starts with the specified
          server, tries to connect to it count-max times, waiting
          interval-secs between each connection attempt, before
          trying the next server in the list (round robin).";
        leaf start-with {
          type enumeration {
            enum first-listed { value 1; }
            enum last-connected { value 2; }
          }
          default first-listed;
          description
           "Specifies which of the application's servers the
            device should start with when trying to connect to
            the application. In the case of newly configured
            application, the first server listed shall be
            considered last-connected.";
        }
        leaf interval-secs {
          type uint8;
          units seconds;
          default 5;
          description
           "Specifies the time delay between connection attempts
            to the same server.  Note: this value differs from the
            periodic-connection's timeout-mins value.";
        }
        leaf count-max {
          type uint8;
          default 3;
          description
           "Specifies the number times the device tries to
```

```
                 connect to a specific server before moving on to
                 the next server in the list (round robin).";
             }
           }
         }
       }
     }


     grouping ssh-config {
        description
         "Provides a reusable grouping for all the ssh config.  This
          is done primarily to enable external modules to reference
          this definition in a &quot;uses&quot; statement.";

         container listen {
           if-feature inbound-ssh;
           description
            "Provides the configuration of the NETCONF server to
             open one or more ports to listen for incoming client
             connections.";
           uses listen-config {
             refine one-or-many/one-port/port {
               default 830;
             }
             refine one-or-many/many-ports/interface/port {
               default 830;
             }
           }
         }

         container call-home {
           if-feature outbound-ssh;
           description
            "Provides the configuration of the NETCONF call-home
             clients to connect to, the overall call-home policy,
             and the reconnect strategy.";
           uses call-home-config {
             augment applications/application {
               container host-keys {
                 description
                  "An ordered listing of the SSH host keys the
                   device should advertise to the application.";
                 list host-key {
                   key name;
                   min-elements 1;
                   ordered-by user;
                   leaf name {
```

```
                     type string {
                       length 1..64; // XXX why this limit?
                     }
                     mandatory true;
                     description
                      "The name of a host key the device should
                       advertise during the SSH key exchange.";
                   }
                 }
               }
             }
           }
         }
      }

     grouping tls-config {
        description
         "Provides a reusable grouping for all the tls config.  This
          is done primarily to enable external modules to reference
          this definition in a &quot;uses&quot; statement.";

         container listen {
           if-feature inbound-tls;
           description
            "Provides the configuration of the NETCONF server to
             open one or more ports to listen for incoming client
             connections.";
           uses listen-config {
             refine one-or-many/one-port/port {
               default 6513;
             }
             refine one-or-many/many-ports/interface/port {
               default 6513;
             }
           }
         }

         container call-home {
           if-feature outbound-tls;
           description
            "Provides the configuration of the NETCONF call-home
             clients to connect to, the overall call-home policy,
             and the reconnect strategy.";
           uses call-home-config;
         }

         /*
          * Objects for deriving NETCONF usernames from X.509
```

```
           * certificates.
           */

          container cert-maps {
            if-feature tls-map-certificates;
            uses x509c2n:cert-to-name;
            description
             "The cert-maps container is used by a 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.";
          }

          /*
           * Objects for deriving NETCONF usernames from TLS
           * pre-shared keys.
           */

          container psk-maps {
            if-feature tls-map-pre-shared-keys;
            description
             "During the TLS Handshake, the client indicates which
              key to use by including a PSK identity in the TLS
              ClientKeyExchange message. On the server side, this
              PSK identity is used to look up an entry in the psk-map
              list. If such an entry is found, and the pre-shared keys
              match, then the client is authenticated. The server uses
              the value from the user-name leaf in the psk-map list as
              the NETCONF username.  If the server cannot find an entry
              in the psk-map list, or if the pre-shared keys do not
              match, then the server terminates the connection.";
            reference
             "RFC 4279: Pre-Shared Key Ciphersuites for Transport Layer
                        Security (TLS)";

            list psk-map {
              key psk-identity;

              leaf psk-identity {
                type string;
                description
                 "The PSK identity encoded as a UTF-8 string. For
                  details how certain common PSK identity formats can
                  be encoded in UTF-8, see section 5.1. of RFC 4279.";
                reference
```

```
                "RFC 4279: Pre-Shared Key Ciphersuites for Transport
                           Layer Security (TLS)";
            }
            leaf user-name {
              type nacm:user-name-type;
              mandatory true;
              description
               "The NETCONF username associated with this PSK
                identity.";
            }
            leaf not-valid-before {
              type yang:date-and-time;
              description
               "This PSK identity is not valid before the given date
                and time.";
            }
            leaf not-valid-after {
              type yang:date-and-time;
              description
               "This PSK identity is not valid after the given date
                and time.";
            }
            leaf key {
              type yang:hex-string;
              mandatory true;
              nacm:default-deny-all;
              description
                "The key associated with the PSK identity";
              reference
                "RFC 4279: Pre-Shared Key Ciphersuites for Transport
                 Layer Security (TLS)";
            }
          }
        }
     }


     /*
      * Configuration data nodes
      */

     container netconf {
       description
        "Top-level container for NETCONF server related
         configuration objects.";

       container ssh {
         if-feature ssh;
```

```
          uses ssh-config;
      }

      container tls {
        if-feature tls;
          uses tls-config;
      }

    }
  }
```

5. Security Considerations

This document defines a YANG modules to configure NETCONF's SSH and TLS transports. Please see the Security Considerations section in those RFCs for transport-specific issues.

6. IANA Considerations

This document registers one URIs in the IETF XML registry [RFC2119]. Following the format in [RFC3688], the following registration is requested to be made.

   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.

This document registers one YANG module in the YANG Module Names registry [RFC6020].

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

7. Acknowledgements

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels ", BCP 14, RFC 2119, March 1997.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF) ", RFC 6020, October 2010.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model ", RFC 6536, March 2012.

[RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013.

[I-D.ietf-netmod-snmp-cfg] Bjorklund, M. and J. Schoenwaelder, "A YANG Data Model for SNMP Configuration", draft-ietf-netmod-snmp-cfg-03 (work in progress), November 2013.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "NETCONF Configuration Protocol", RFC 6241, June 2011.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.

[I-D.ietf-netconf-rfc5539bis] Badra, M., Luchuk, A., and J. Schoenwaelder, "Using the NETCONF Protocol over Transport Layer Security (TLS) ", draft-ietf-netconf-rfc5539bis-04 (work in progress), October 2013.

8.2. Informative References

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

Appendix A. Example: SSH

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

    <listen>
      <port>831</port>
    </listen>
```

```
        <call-home>
          <applications>
            <application>
              <name>config-mgr</name>
              <description>
                 This entry requests the device to periodically
                 connect to the Configuration Manager application
              </description>
              <servers>
                 <server>
                    <address>config-mgr1.example.com</address>
                 </server>
                 <server>
                    <address>config-mgr2.example.com</address>
                 </server>
              </servers>
              <connection-type>
                <periodic>
                  <timeout-mins>5</timeout-mins>
                  <linger-secs>10</linger-secs>
                </periodic>
              </connection-type>
              <reconnect-strategy>
                 <start-with>last-connected</start-with>
                 <interval-secs>10</interval-secs>
                 <count-max>3</count-max>
              </reconnect-strategy>
              <host-keys>
                 <host-key>
                    <name>ssh_host_key_cert</name>
                 </host-key>
                 <host-key>
                    <name>ssh_host_key_cert2</name>
                 </host-key>
              </host-keys>
            </application>
          </applications>
        </call-home>

      </ssh>
    </netconf>
```

Appendix B. Example: TLS

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

        <listen>
```

```
          <interface>
            <address>192.0.2.1</address>
            <port>6514</port>
          </interface>
        </listen>

        <call-home>
          <applications>
            <application>
              <name>log-monitor</name>
              <description>
                 This entry requests the device to maintain a
                 persistent connect to the Log Monitor application
              </description>
              <servers>
                 <server>
                    <address>log-monitor1.example.com</address>
                 </server>
                 <server>
                    <address>log-monitor2.example.com</address>
                 </server>
              </servers>
              <connection-type>
                <persistent>
                  <keep-alives>
                    <interval-secs>5</interval-secs>
                    <count-max>3</count-max>
                  </keep-alives>
                </persistent>
              </connection-type>
              <reconnect-strategy>
                 <start-with>first-listed</start-with>
                 <interval-secs>10</interval-secs>
                 <count-max>4</count-max>
              </reconnect-strategy>
            </application>
          </applications>
        </call-home>

        <cert-maps>
          <!-- Use a subject alt name field of a specific
               certificate as the NC username. -->
          <cert-to-name>
            <id>1</id>
            <fingerprint>11:0A:05:11:00</fingerprint>
            <map-type>x509c2n:san-any</map-type>
          </cert-to-name>
          <!-- Map a specific certificate to the NC username
```

```
              'Joe Cool'. -->
          <cert-to-name>
            <id>2</id>
            <fingerprint>11:0A:05:11:00</fingerprint>
            <map-type>x509c2n:specified</map-type>
            <name>Joe Cool</name>
          </cert-to-name>
        </cert-maps>

        <psk-maps>
          <psk-map>
            <psk-identity>a8gc8]klh59</psk-identity>
            <user-name>admin</user-name>
            <not-valid-before>2013-01-01T00:00:00Z</not-valid-before>
            <not-valid-after>2014-01-01T00:00:00Z</not-valid-after>
          </psk-map>
        </psk-maps>

      </tls>
    </netconf>
```


Appendix C. Change Log

C.1. 00 to 01

- Restructured YANG module slightly, to provide groupings useful to the ZeroTouch draft.

Appendix D. Open Issues

- NETCONF implementations typically have config parameters such as session timeouts or hello timeouts. Shall they be included in this model?

- Do we need knobs to enable/disable call-home without the need to remove all the call-home client configuration?

- Do we need something equivalent to the host-keys in the TLS configuration subtree?

Authors' Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Juergen Schoenwaelder
Jacobs University Bremen

EMail: j.schoenwaelder@jacobs-university.de

NETCONF Working Group K. Watsen
Internet-Draft S. Hanna
Intended status: Standards Track Juniper Networks
Expires: August 05, 2014 J. Clarke
Cisco Systems
M. Abrahamsson
T-Systems
February 2014

Zero Touch Provisioning for NETCONF Call Home (ZeroTouch)
draft-kwatsen-netconf-zerotouch-01

Abstract

This draft presents a technique for how to establish a secure NETCONF connection between a newly deployed networking device, configured with just its factory default settings, and the new owner's Network Management System (NMS).

Status of This Memo

This Internet-Draft will expire on August 05, 2014.

Copyright Notice

Table of Contents

1. 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. Introduction

The solution presented herein is designed to support the NETCONF configuration protocol [RFC6241]. This is achieved by leveraging the recently standardized call home mechanisms for SSH [REVERSE-SSH] and TLS [RFC5539bis].

A fundamental business requirement is to reduce operational costs where possible. Deploying new devices is typically one of the largest costs in running the network, as sending trained specialists to each site is both cost prohibitive and doesn't scale.

Both networking vendors and standard bodies have tried to address this issue, with varying levels of success. For instance, the Broadband Forum TR-069 specification [TR069] relies on DHCP for NMS discovery, but this only works in environments where the DHCP server can be configured, which isn't the case when the device is connected to an ISP's network. In another example, some network vendors have enabled their devices to load an initial configuration from removable storage media (e.g., a USB flash drive), but not all devices have such ports.

The solution presented herein attempts to addresses the evaluation criteria outlined by the draft "Configuring Security Parameters in Small Devices" [draft-hanna-zeroconf-seccfg-00]: security, flexibility, ease of use, and device cost. More specifically:

o Security

Security is fundamental to any automated discovery solution, especially one that bootstraps the parameters used to secure a device's connection to its NMS. Consistent with [RFC3365], security is a required aspect of ZeroTouch. Every ZeroTouch implementation is sure to be secure.

o Flexibility

ZeroTouch is designed to support a wide variety of deployments, including cases where the device is connected to a network without administrative control of the local DHCP and DNS servers, where the device is connected to an untrusted network, or deployed behind a gateway that dynamically translates its network address (e.g., NAT). Special consideration is also provided for devices that are on a network with no public Internet access.

o Ease of Use

Ultimately, the success of the solution depends on the ability for presumably non-technical personnel to be able to complete the installation by themselves. To this end, it is envisioned that installers only need to connect the device to a wired or wireless network and a power source and wait for the device to indicate success. ZeroTouch also attempts to not be overly complicated for NMS administrators.

- o Device Cost

  For vendors of devices with already slim margins, such as consumer-oriented devices, a significant concern is the cost of goods. Fortunately, the solution presented in this draft requires minimal additional components. Additionally, the development effort doesn't seem exorbitant, though that may vary by vendor and their circumstances.

3. Actors and Roles

3.1. Device

The device is the networking entity that initiates Zero Touch. Whenever a device boots with its factory default settings, it initiates ZeroTouch with the goal of finding a Configlet with which it can use to configure itself. Once a Configlet is found, the device initializes its configuration using that Configlet and then exits ZeroTouch. Since the Configlet configures the device to "call home" upon entering its normal operating mode, the device immediately begins trying to establish a reverse-SSH or reverse-TLS connection, as specified by the Configlet.

3.2. Configlet

A Configlet is an XML file, containing specific YANG-defined configuration, that has been signed by a trusted signer known to the device (e.g., the device's manufacturer).

The Configlet data-model, defined by the YANG module in this document (see Section 3.2), is just enough to configure a local user account and either reverse-SSH or reverse-TLS. More specifically, this data-model is a subset of what's defined in ietf-system and ietf-netconf-server YANG models. This focused data-model is consistent with the common use-case of having the NMS push a full configuration to a device when it calls home.

The signature on the Configlet is enveloped, meaning that the signature is contained inside the XML file itself. The signature block also contains the X.509 certificate of the Configlet Signer and its chain of trust.

Once a device authenticates the signature on a Configlet and matches the unique identifier (e.g., serial number) within the Configlet, it merges the configuration contained in the Configlet into its running datastore.

3.3. Configlet Signer

A Configlet Signer is the entity authorized by the device manufacturer to sign Configlets for its devices (note: this may be the device vendor itself).

A Configlet Signer MUST provide a user-facing service enabling the creation of a Configlet with user-specific deployment values, using the YANG schema defined in Section 3.2. This document does not specify what form this interface must take, so it is the Configlet Signer's discretion if it is a web page, a REST API, or something else.

A Configlet Signer MUST ensure that the end-user is the rightful owner of the device containing the unique identifier value to be put into the Configlet. How a Configlet Signer ensures this is outside the scope of this document, but it is envisioned that the vendor would provide a secured interface for its trusted Configlet Signers to use.

A Configlet Signer MUST have an X.509 certificate with Key Usage capable of signing data (digitalSignature) and be signed by a certificate authority having a chain of trust leading to a trust anchor known to the devices loading its Configlets. The Configlet Signer MUST possess all intermediate certificates leading to its trust anchor.

When a Configlet Signer signs a Configlet, it attaches both the signature and the chain of X.509 certificates, including its own, but not necessarily including the trust anchor's certificate. This chain of certificates is needed so a device can validate a Configlet using only the Configlet Signer trust anchors known to it.

A Configlet Signer does not need to retain the Configlet after signing it; it is expected that either the end-user or the Configlet Signer will convey the signer Configlet to a Configuration Server where it will be hosted.

3.4. Configuration Server

A Configuration Server is the entity hosting configurations that can be downloaded over a network. Configuration Servers are known to devices in the form of a URI, to which a device appends the fingerprint of its entity certificate (see Section 4.1 for details). For instance, if the URI were:

```
https://example.com?id=
scp://user@zerotouch.example.com/configlets/
ftp://example.com/zerotouch/configlets/
```

then the device would try to access:

```
https://example.com?id=<fingerprint>
scp://user@zerotouch.example.com/configlets/<fingerprint>
ftp://example.com/zerotouch/configlets/<fingerprint>
```

where the fingerprint is generated using the SHA-256 algorithm over the device’s entity certificate. For instance, using OpenSSL’s command line tool: ‘openssl dgst -sha256 <entity certificate>‘ (see Section 4.1).

The Configuration Server is expected to be able to map the fingerprint to a device-specific unique identifier (e.g., serial number), which is the value contained in the Configlet it is hosting. How the Configuration Server does this mapping is outside the scope of this document, but it is envisioned that the vendor would provide a secured interface for its trusted Configuration Servers to use.

Configuration Servers do not need to use encryption, since the Configlets themselves are immutable to tampering, due to being signed. However, for confidentiality reasons, it is RECOMMENDED to use encryption, so adversaries cannot see the Configlet’s otherwise clear-text content, from which they can learn some details about the device’s internal deployment.

If a Configuration Server uses X.509-based encryption, then its X.509 certificate MUST have a chain of trust to a trust anchor known to devices. The Configuration Server MUST possess all the intermediate certificates leading to a trust anchor.

When a Configuration Server negotiates encryption with the device, it provides the chain of X.509 certificates, including its own, but not necessarily including the trust anchor’s certificate. Devices need the chain of certificates to be passed so they can validate the server using only a minimal list of Configuration Server trust anchors.

Configuration Server’s SHOULD automatically expire Configlets after some user-specified amount of time.

In order to facilitate troubleshooting and auditing, the Configuration Server SHOULD record into a log a record of the various Configlet download requests. This draft does not define what information should be kept or for how long.

3.5. Network Management System (NMS)

The NMS is the ultimate destination of ZeroTouch for a device. It is the NMS's network address configured in the Configlet. The device will connect to the NMS using either a reverse-SSH or reverse-TLS, as configured by the Configlet loaded.

In order to authenticate the device, the NMS MUST possess the X.509 certificate for the trust anchor leading to the device's entity certificate. The NMS uses this certificate to validate the server-certificate the device presents during SSH or TLS transport negotiation.

The NMS SHOULD also validate that the unique identifier (e.g., serial number), within the "Common Name" field of the device's X.509 certificate, is an identity that the NMS is expecting, and not another device having the same device type. In order for the NMS to know the unique identifiers for devices shipped directly to their destinations, it may be necessary for the device manufacturer to provide the unique identifiers along with other shipping or billing information. This draft not specify a format for this information exchange.

In addition to authenticating the device, the NMS must also authenticate itself to the device. How this is done is protocol specific. For reverse-SSH, the NMS needs to know the information configured on the device by the Configlet it loaded, specifically the name of a local user account and the necessary credentials configured for the account (i.e., password or the private-half of a SSH host key). For reverse-TLS, the NMS must present a client certificate. Presumably the NMS has been configured with the information used when the Configlet was created.

4. Device Boot Sequence

4.1. Precondition

Devices supporting ZeroTouch MUST support either reverse-SSH or reverse-TLS, and MAY support both. In either case, the device MUST present an X.509-based certificate encoding a unique device identifier (e.g. serial number) and a public key, both signed by a trusted certificate authority. This certificate is the "entity certificate" in the diagram below. This certificate is needed in order for the NMS to positively authenticate a device. For reverse-SSH, this certificate requirement constrains the SSH host key algorithms the device is allowed to to use to those defined in [RFC6187].

The unique identifier MUST be something that is both known to the device and easily tracked through labels affixed to the device as well as the box it is packaged in. A device's serial number is commonly treated this way and would be suitable for this purpose.

The device MUST possess the private key matching the public key encoded in the entity certificate. Ideally, the private key SHOULD be generated and protected by a tamper-resistant cryptographic processor, as this provides the greatest assurance that the private key is known to no one, including the device's manufacturer.

The entity certificate MUST be signed by a certificate authority having a chain of trust to a well-known trust anchor. The device MUST also possess the X.509 certificates for any intermediate CAs leading to the trust anchor. During SSH or TLS transport setup, the device will send both its entity and any intermediate certificates so the NMS can verify the certificate path using only the well-known trust anchor.

Since the entity certificate is to be used for SSH and TLS connections, its Key Usage, if set, SHOULD have the digitalSignature, keyEncipherment, and keyAgreement bits set.

In order for a device to know where it can obtain a Configlet, it MUST have two sets of URIs, identifying resources where it can obtain Configlets. One set contains secure schemes (e.g. https://, scp://) and the other contains insecure schemes (e.g., http://, ftp://). These URIs point to the "Configuration Servers" in the diagram below.

In order for a device to use a URI with a secure scheme, devices MUST possess a trust anchor certificate that it can use to authenticate the Configuration Server with. As each Configuration Server may use a different trust anchor, this generalizes to a list of Configuration Server trust anchor certificates. These trust anchors MAY include broadly recognized certificate authorities, such as the certificates packaged with web browsers.

In order for a device to authenticate Configlets, it MUST have a trust anchor for the CA that signed the Configlet. The CA used to sign Configlets is called a "Configlet Signer" in the diagram below. In order to enable Configlets to be signed by different CAs, the device MAY have either a list of trust anchors, or a single trust anchor that delegates Configlet signing trust to other CAs. The diagram below shows a list of Configlet Signer trust anchors only because it is the more flexible option.

Devices SHOULD ensure that all its trust anchor certificates, including those for the Configlet Signer and Configuration Server, are protected from external modification. It is for this reason that the diagram below shows them in immutable storage. However, it may be necessary to update these certificates over time (e.g., the vendor wants to delegate trust to a new CA). Therefore, devices MAY update these trust anchors when needed through a verifiable process, such as a software upgrade using signed software images.

Devices SHOULD ensure that the certificates for its trust anchors are protected from external modification, specifically the Configlet Signer and Configuration Server X.509 certificates. It is for this reason that the diagram below shows them in immutable storage. The certificates for the device’s trust anchors MAY be updated along with a standard software image upgrade.

Device State Precondition

```
+------------------------------------------------------------------+
|                            <device>                              |
|                                                                  |
|     +-------------------------------------------------------+    |
|     |                <immutable storage>                    |    |
|     |                                                       |    |
|     |  list of Configlet Signer trust anchor certificates   |    |
|     |  list of Configuration Server trust anchor certs      |    |
|     +-------------------------------------------------------+    |
|                                                                  |
|   +----------------------------------------------------------+   |
|   |                      <other storage>                     |   |
|   |                                                          |   |
|   |  two sets of Configuration Server URIs                   |   |
|   |  device entity & associated intermediate certificate(s)  |   |
|   +----------------------------------------------------------+   |
|                                                                  |
|                    +----------------------+                      |
|                    |  <crypto processor>  |                      |
|                    |                      |                      |
|                    |  device private key  |                      |
|                    +----------------------+                      |
|                                                                  |
+------------------------------------------------------------------+
```

4.2. Runtime Operation

Whenever a device boots with its factory default settings, it initiates ZeroTouch with the goal of finding a configuration with which it can use to configure itself.

The device MUST first initialize its networking as per its default factory configuration. This SHOULD result in the dynamic assignment of an IP address, subnet or prefix, gateway, and a DNS server.

While initializing its networking, the device MAY receive some additional URIs for where a software image or configuration can be downloaded. This draft does not define how such URIs MAY be exchanged, for instance, using DHCP options.

If, while initializing its networking, the device receives software image URIs, it MAY download and install the software image only if the image is protected from modification (e.g., the image is signed) and the device is able to verify its integrity. The device SHOULD try URIs with secure schemes before URIs with insecure schemes (e.g., scp:// before ftp://). If the device needs to reboot to activate the new software image, it MUST do so with its default factory configuration set so that ZeroTouch will run again when the device comes back up.

If, while initializing its networking, the device receives configuration URIs, each URI SHOULD be appended to one of the device’s two sets of Configuration Server URIs, depending on if the URI’s scheme is secure or not. URIs added this way MUST remain distinguishable from those URIs the device was shipped with, for reasons discussed in Section 4.2.

Before trying any of the Configuration Server URIs, the device SHOULD first try to load a configuration through local means that assert physical presence. For instance, a removable USB flash drive or near-field communication mechanism. Configurations obtained through an assertion of physical presence do not have to be signed or contain the device’s unique identifier (e.g., serial number). If a Configlet is found, the device MUST use it without trying any of the Configuration Server URIs.

If a configuration was not found via physical presence, the device then iterates over its two sets of Configuration Server URIs. The device MUST first try all the URIs from the set having secure schemes before trying any of the URIs from the set having insecure schemes. For each URI, until a usable configuration is found and successfully loaded, the device attempts to download a configuration from the URI. If the URI uses a secure scheme (e.g., https), the device MUST validate the Configuration Server’s certificate using one of its Configuration Server trust anchors. If the device is unable to verify the server’s certificate, the device MUST skip that URI. If the device reaches the end of all its URIs without finding a configuration, it MAY continue its normal boot sequence using its factory default configuration.

When the device is accessing a Configuration Server URI that it was shipped with (i.e., not discovered while initializing its networking, it MUST do so by appending the fingerprint of its entity certificate to the URI’s string, as described in the Section 3.4. For URIs discovered while initializing its networking, the device MAY try both the raw URI as well as the permutation of it using its fingerprint.

The Configuration Server’s response MAY be anything allowed by the given URI’s scheme. For instance, if the scheme is HTTP-based, the Configuration Server MAY return an HTTP redirect. In this way, a vendor’s Configuration Server service may allow the device owner to redirect the device to a Configuration Server running in their network.

If the Configuration Server returns a Configlet, the device MUST first verify it before use. Configlet verification entails both verifying the Configlet’s signature using the device’s list of Configlet Signer trust anchors, and also verifying that the unique identifier (e.g., serial number) within the Configlet matches the device’s unique identifier.

Once a Configlet is authenticated, the device merges the Configlet’s contents into its running configuration and then exits ZeroTouch. Since the Configlet configures the device to "call home," upon entering its normal operating mode, the device immediately begins trying to establish a reverse-SSH or reverse-TLS connection, as specified by the Configlet.

If configured to establish a reverse-SSH connection, the the device MUST use its entity and associated intermediate X.509 certificates as its host key per RFC 6187 [RFC6187]. If configured to use reverse-TLS, the device MUST use its entity and associated intermediate X.509 certificates as its server-side certificate for the TLS connection.

In order to facilitate troubleshooting, the device SHOULD record into a log information relating to its stepping through the ZeroTouch sequence of steps. This draft does not define any specific log messages, for instance, for Syslog or SNMP.

ZeroTouch Sequence Diagram

```
          DEVICE     LOCAL DHCP         CONFIGLET         NMS
             |         SERVER            SERVERS           |
             |             |                  |            |
             |------------>|                  |            |
             | Lease IP    |                  |            |
             |             |                  |            |
             |------------------------------->|            |
             | Iterate until Configlet found  |            |
             |             |                  |            |
             |             |                  |            |
             |--------------------+           |            |
             | Validate Configlet |           |            |
             |<-------------------+           |            |
             |             |                  |            |
             |             |                  |            |
             |--------------------------------------------->|
             | reverse-SSH or reverse-TLS     |            |
             |             |                  |            |
             |             |                  |            |
```

5. Configlets

5.1. Data Model

The Configlet’s data is modeled after the data models provided by draft-ietf-netmod-system-mgmt and draft-kwatsen-netconf-server. These data models are used to configure a local user account and either reverse-SSH or reverse-TLS. Networking is not included in the Configlet data model as it is expected that the device will by assigned a dynamic address by the network and that it will use this address both when connecting to a Configuration Server and later the NMS.

From draft-ietf-netmod-system-mgmt, the data model for user authentication has the following structure:

```
+--rw system
   +--rw authentication
      +--rw user-authentication-order*   identityref
      +--rw user* [name]
         +--rw name        string
         +--rw password?   crypt-hash
         +--rw ssh-key* [name]
            +--rw name         string
            +--rw algorithm    string
            +--rw key-data     binary
```

From draft-kwatsen-netconf-server, the data model for reverse-SSH has the following structure:

```
+--rw netconf
   +--rw ssh {ssh}?
      +--rw listen {inbound-ssh}?
      |  +--rw (one-or-many)?
      |     +--:(one-port)
      |     |  +--rw port?        inet:port-number
      |     +--:(many-ports)
      |        +--rw interface* [address]
      |           +--rw address    inet:ip-address
      |           +--rw port?      inet:port-number
      +--rw call-home {outbound-ssh}?
         +--rw applications
            +--rw application* [name]
               +--rw name                  string
               +--rw description?          string
               +--rw servers
               |  +--rw server* [address]
               |     +--rw address    inet:host
               |     +--rw port?      inet:port-number
               +--rw connection-type
               |  +--rw (connection-type)?
               |     +--:(persistent-connection)
               |     |  +--rw persistent
               |     |     +--rw keep-alives
               |     |        +--rw interval-secs?   uint8
               |     |        +--rw count-max?       uint8
               |     +--:(periodic-connection)
               |        +--rw periodic
               |           +--rw timeout-mins?   uint8
               |           +--rw linger-secs?    uint8
               +--rw reconnect-strategy
               |  +--rw start-with?      enumeration
               |  +--rw interval-secs?   uint8
               |  +--rw count-max?       uint8
               +--rw host-keys
                  +--rw host-key* [name]
                     +--rw name    string
```

Also from draft-kwatsen-netconf-server, the data model for reverse-TLS has the following structure:

```
   +--rw netconf
      +--rw tls {tls}?
         +--rw listen {inbound-tls}?
         |  +--rw (one-or-many)?
         |     +--:(one-port)
         |     |  +--rw port?        inet:port-number
         |     +--:(many-ports)
         |        +--rw interface* [address]
         |           +--rw address    inet:ip-address
         |           +--rw port?      inet:port-number
         +--rw call-home {outbound-tls}?
         |  +--rw applications
         |     +--rw application* [name]
         |        +--rw name                  string
         |        +--rw description?          string
         |        +--rw servers
         |        |  +--rw server* [address]
         |        |     +--rw address    inet:host
         |        |     +--rw port?      inet:port-number
         |        +--rw connection-type
         |        |  +--rw (connection-type)?
         |        |     +--:(persistent-connection)
         |        |     |  +--rw persistent
         |        |     |     +--rw keep-alives
         |        |     |        +--rw interval-secs?   uint8
         |        |     |        +--rw count-max?       uint8
         |        |     +--:(periodic-connection)
         |        |        +--rw periodic
         |        |           +--rw timeout-mins?   uint8
         |        |           +--rw linger-secs?    uint8
         |        +--rw reconnect-strategy
         |           +--rw start-with?      enumeration
         |           +--rw interval-secs?   uint8
         |           +--rw count-max?       uint8
         +--rw cert-maps {tls-map-certificates}?
         |  +--rw cert-to-name* [id]
         |     +--rw id             uint32
         |     +--rw fingerprint    x509c2n:tls-fingerprint
         |     +--rw map-type       identityref
         |     +--rw name           string
         +--rw psk-maps {tls-map-pre-shared-keys}?
            +--rw psk-map* [psk-identity]
               +--rw psk-identity        string
               +--rw user-name           nacm:user-name-type
               +--rw not-valid-before?   yang:date-and-time
               +--rw not-valid-after?    yang:date-and-time
               +--rw key                 yang:hex-string
```

5.2. Signature

Configlets obtained over the network MUST be signed using the W3C standard "XML Signature Syntax and Processing" [XMLSIG]. The entire contents of the Configlet MUST be signed. The signature block must also include the Configlet Signer's certificate and any intermediate certificates leading to a Configlet Signer trust anchor.

A signed Configlet example is in the Appendix.

5.3. YANG Module

Following is the YANG module for the Configlet:

```
module ietf-netconf-zerotouch {

  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-zerotouch";
  prefix "nczerotouch";

  //import ietf-system {
  //    prefix ncsystem;
  //}

  import ietf-netconf-server {
      prefix ncserver;
  }

  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: Bert Wijnen
              <mailto:bertietf@bwijnen.net>

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

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

    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.";
     // RFC Ed.: replace XXXX with actual RFC number and
     // remove this note

     // RFC Ed.: please update the date to the date of publication

     revision "2014-01-24" {
       description
        "Initial version";
       reference
        "RFC XXXX: A YANG Data Model for NETCONF ZeroTouch Configlet";
     }

     typedef crypt-hash {
       type string {
         pattern
           '$0$.*'
         + '|$1$[a-zA-Z0-9./]{1,8}$[a-zA-Z0-9./]{22}'
         + '|$5$(rounds=\d+$)?[a-zA-Z0-9./]{1,16}$[a-zA-Z0-9./]{43}'
         + '|$6$(rounds=\d+$)?[a-zA-Z0-9./]{1,16}$[a-zA-Z0-9./]{86}';
       }
       description
         "The crypt-hash type is used to store passwords using
          a hash function.  The algorithms for applying the hash
          function and encoding the result are implemented in
          various UNIX systems as the function crypt(3).

          A value of this type matches one of the forms:

            $0$<clear text password>
            $<id>$<salt>$<password hash>
            $<id>$<parameter>$<salt>$<password hash>

          The '$0$' prefix signals that the value is clear text.  When
          such a value is received by the server, a hash value is
          calculated, and the string '$<id>$<salt>$' or
          $<id>$<parameter>$<salt>$ is prepended to the result.  This
          value is stored in the configuration data store.
```

```
         If a value starting with '$<id>$', where <id> is not '0', is
         received, the server knows that the value already represents a
         hashed value, and stores it as is in the data store.

         When a server needs to verify a password given by a user, it
         finds the stored password hash string for that user, extracts
         the salt, and calculates the hash with the salt and given
         password as input.  If the calculated hash value is the same
         as the stored value, the password given by the client is
         accepted.

         This type defines the following hash functions:

           id | hash function | feature
           ---+---------------+-------------------
            1 | MD5           | crypt-hash-md5
            5 | SHA-256       | crypt-hash-sha-256
            6 | SHA-512       | crypt-hash-sha-512

         The server indicates support for the different hash functions
         by advertising the corresponding feature.";
      reference
        "IEEE Std 1003.1-2008 - crypt() function
         Wikipedia: http://en.wikipedia.org/wiki/Crypt_(C)
         RFC 1321: The MD5 Message-Digest Algorithm
         FIPS.180-3.2008: Secure Hash Standard";
    }


    container configlet {
      description
       "Top-level container for ZeroTouch configuration objects.";

      container system {
        // no way to use top-level "netconf" container?
        container authentication {

          list user {
            key name;
            description
              "The list of local users configured on this device.";

            leaf name {
              type string;
              description
               "The user name string identifying this entry.";
            }
            leaf password {
```

```
              type crypt-hash;
              description
                "The password for this entry.";
            }
            list ssh-key {
              key name;
              description
                "A list of public SSH keys for this user.";
              reference
                "RFC 4253: The Secure Shell (SSH) Transport Layer
                           Protocol";

              leaf name {
                type string;
                description
                  "An arbitrary name for the ssh key.";
              }
              leaf algorithm {
                type string;
                mandatory true;
                description
                  "The public key algorithm name for this ssh key.

                   Valid values are the values in the IANA Secure Shell
                   (SSH) Protocol Parameters registry, Public Key
                   Algorithm Names";
                reference
                  "IANA Secure Shell (SSH) Protocol Parameters
                   registry, Public Key Algorithm Names";
              }
              leaf key-data {
                type binary;
                mandatory true;
                description
                  "The binary key data for this ssh key.";
              }
            }
          }
        }
      }

      container netconf-server {
        // no way to use top-level "netconf" container?

        container ssh {
          uses ncserver:ssh-config;
            // no way to disable "listen" container?
        }
```

```
         container tls {
           uses ncserver:tls-config;
             // no way to disable "listen" container?
         }
       }


     }
   }
```

6. Security Considerations

It is not possible to substitute a Configlet created for a different device, since devices assert that the Configlet contains their unique identifier (e.g., serial number).

It is possible to substitute a Configlet created for a device with a different Configlet created for the same device. Generally, unless imposed by the Configlet Signers, there is no limit to the number of Configlets that may be generated for a given device. This could be resolved, in part, by placing a timestamp into the Configlet and ensuring devices do not load Configlets older than some amount, but this requires the devices have an accurate clock when validating a Configlet and for Configlet Signers to not sign a Configlet when another Configlet is still active.

Confidentiality of Configlets loaded over a network is only assured when the device uses a secure networking scheme and validates the Configuration Server’s certificate.

Confidentiality is further provided by using the fingerprint of the device’s entity certificate when doing a Configuration Server lookup, as it is not guessable and thus makes it nearly impossible for an adversary to lookup.

This draft allows devices to try alternate means to load a Configlet before trying the network, so long as they assert physical presence. For instance, a removable USB drive or a near-field communication mechanism. Further, this draft does not require Configlets to be signed, if loaded via a mechanism that asserts physical presence. or require those Configlets to have the device’s unique identifier value set. All of these relaxations in Security are deemed acceptable because physical presence should only be accessible to trusted parties.

This draft allows devices to use insecure schemes when doing a Configuration Server lookup. This is deemed acceptable because the Configlet is tamper-proof, due to being signed, only confidentiality is lost.

This draft entails the device having an X.509 certificate that is used by the NMS to authenticate the device. This certificate and every certificate in the chain leading to the well known trust anchor, should have a expiration date greater than the device's useful life expectancy. Given the long-lived nature of these device certificates, it is paramount to use a strong key length (e.g., 512-bit ECC). Configlet Signers should deploy Online Certificate State Protocol (OCSP) responders or CRL Distribution Points (CDP) to revoke certificates in case necessary.

This draft mentions using the device's serial number as its unique identifier in its entity 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. To address this concern, the certificate could contain the hash of the serial number instead, which the NMS could also compute, but doing so is much less intuitive and raises questions if it is just security through obscurity.

It is paramount the device manufacturer ensures the integrity of the device's list of trust anchors. It should not be possible for anyone other than the manufacturer be able to modify the list of trust anchors. One way to achieve this to sign the list of trust anchors with a private key known only to the manufacturer, and for the matching public key to be stored on tamper-resistant read-only media.

7. IANA Considerations

None

8. Acknowledgements

The authors would like to thank Russ Mundy and Wes Hardaker for brainstorming the solution presented in this draft with us during the IETF 87 meeting in Berlin.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels ", BCP 14, RFC 2119, March 1997.

[RFC3365] Schiller, J., "Strong Security Requirements for Internet Engineering Task Force Standard Protocols ", RFC 3365, August 2002.

[RFC4252] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Authentication Protocol ", RFC 4252, January 2006.

[RFC5539bis] Badra, M. and A. Luchuk, "Using the NETCONF Protocol over Transport Layer Security (TLS) ", RFC 5539, March 2011.

[RFC6187] Igoe, K. and D. Stebila, "X.509v3 Certificates for Secure Shell Authentication ", RFC 6187, March 2011.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "NETCONF Configuration Protocol", RFC 6241, June 2011.

[REVERSE-SSH] Watsen, K., "Reverse SSH", June 2013.

[XMLSIG] , "XML Signature Syntax and Processing", June 2008.

9.2. Informative References

[TR069] The Broadband Forum, ., "TR-069 Amendment 3, CPE WAN Management Protocol ", November 2010.

[draft-hanna-zeroconf-seccfg-00] Hanna, ., "Configuring Security Parameters in Small Devices ", January 2002.

Appendix A. Examples

A.1. Signed Configlet

This example illustrates a Configlet configuring both a local user account and reverse-SSH. This Configlet includes both the Configlet Signer’s certificate as well as an Intermediate certificate. Note that ’\’ characters have been added for formatting reasons.

```
<?xml version="1.0"?>
<configlet xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-zerotouch">
  <!-- from ietf-system.yang -->
  <system>
```

```
       <authentication>
         <user>
           <name>admin</name>
           <ssh-key>
             <name>admin’s rsa ssh host-key</name>
             <algorithm>ssh-rsa</algorithm>
             <key-data>AAAAB3NzaC1yc2EAAAADAQABAAABAQDeJMV8zrtsi8CgEsRC
             jCzfve2m6zD3awSBPrh7ICggLQvHVbPL89eHLuecStKL3HrEgXaI/O2Mwj
             E1lG9YxLzeS5p2ngzK61vikUSqfMukeBohFTrDZ8bUtrF+HMLlTRnoCVcC
             WAw1lOr9IDGDAuww6G45gLcHalHMmBtQxKnZdzU9kx/fL3ZS5G76Fy6sA5
             vg7SLqQFPjXXft2CAhin8xwYRZy6r/2N9PMJ2Dnepvq4H2DKqBIe340jWq
             EIuA7LvEJYql4unq4Iog+/+CiumTkmQIWRgIoj4FCzYkO9NvRE6fOSLLf6
             gakWVOZZgQ8929uWjCWlGlqn2mPibp2Go1</key-data>
           </ssh-key>
         </user>
       </authentication>
     </system>
     <!-- from ietf-netconf-server.yang -->
     <netconf-server>
       <ssh>
         <call-home>
           <applications>
             <application>
               <name>config-mgr</name>
               <description>
                This entry requests the device to periodically
                connect to the Configuration Manager application
             </description>
               <servers>
                 <server>
                   <address>config-mgr1.example.com</address>
                 </server>
                 <server>
                   <address>config-mgr2.example.com</address>
                 </server>
               </servers>
               <connection-type>
                 <periodic>
                   <timeout-mins>5</timeout-mins>
                   <linger-secs>10</linger-secs>
                 </periodic>
               </connection-type>
               <reconnect-strategy>
                 <start-with>last-connected</start-with>
                 <interval-secs>10</interval-secs>
                 <count-max>3</count-max>
               </reconnect-strategy>
               <host-keys>
```

```
                 <host-key>
                   <name>ssh_host_key_cert</name>
                 </host-key>
                 <host-key>
                   <name>ssh_host_key_cert2</name>
                 </host-key>
               </host-keys>
             </application>
           </applications>
         </call-home>
       </ssh>
     </netconf-server>
     <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
       <SignedInfo>
         <CanonicalizationMethod
           Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
         <SignatureMethod
           Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <Reference>
           <Transforms>
             <Transform
               Algorithm=\
               "http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
           </Transforms>
           <DigestMethod
             Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
           <DigestValue>2xlFdlVifb1snGBLJuEZYrLjSUQ=</DigestValue>
         </Reference>
       </SignedInfo>
       <SignatureValue>\
   HUx3S7TZXGJGUhazWGRSB9CBMZ0T+tTrB1fOnTcKi9wU4UOnSw5KMWDvOVwc6ldM
   UIOJIuJigWhSkn+VvWSWz6qy7LTYIywNcxDyghMvmMXfoRXETpL+qCDxribMi4VW
   mVhEw1oe83kJt7W/0DJUE7FFKRUhPjy9EgxpQX/7WdKSK+4f2uYkSpq2UumW3DIU
   LeK9vNRVQBbhmcF3zZWANmwKH5V4WeQimwWE497AeSYWgSImSetADI0NvvXfBZjx
   JqzFEaYLNz8IB0ZVY+w14s1RZbN7YmxhN1R3q52wWvHjR2SylR/Z5BpIhYoDeKoD
   HMQMf3HZL06Hm5S8r8rgGg==</SignatureValue>
       <KeyInfo>
         <X509Data>
           <X509Certificate>\
   MIIFKjCCBBKgAwIBAgIBAjANBgkqhkiG9w0BAQsFADAwMRMwEQYDVQQKFApUUE1f
   VmVuZG9yMRkwFwYDVQQDFBBKdW5pcGVyX1hYWFhYX0NBMB4XDTEzMTAyMDE2MjIx
   MFoXDTE0MTAyMDE2MjIxMFowKzETMBEGA1UEChQKVFBNX1ZlbmRvcjEUMBIGA1UE
   AxQLY2hpcF8wMDAwMDEwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQDf
   4hyWqFsf801sZYJQBJ0PB4cHmlnPNOs9pv3QCCB1PzlYhfcDOygVmqhzZjPY+t7q
   ZTjPs/E8n5X4dd0DkR80uc4MWmzc40Pz2HAW6GQ2mo+eUYzXUqQFbi3EkqrzddZk
   gRi6vuadMkAcJH8ugYR+cbw/LlpXhIy2A5fUh4JP7Y9l1wABTbK8eGhF9cvGxBYR
   +KqZJycoV6aaIvD/0NO1CNSaGeAJXXxXWoRF5E6HVKsolTHPPdi+40BmYrCuuWy6
   1ybCIP5uZZ7Oza4j0n/fPb6SEqEa0I1zUEWlFQMZYsBClNY5TzWHNgQ5dPJO2qgx
```

```
PONwnLIsx46DlAzlpFpXAgMBAAGjggJSMIICTjAMBgNVHRMBAf8EAjAAMIGTBgNV
HSABAf8EgYgwgYUwgYIGC2CGSAGG+EUBBy8BMHMwOQYIKwYBBQUHAgEWLWh0dHA6
Ly93d3cudmVyaXNpZ24uY29tL3JlcG9zaXRvcnkvaW5kZXguaHRtbDA2BggrBgEF
BQcCAjAqGihUQ1BBIFRydXN0ZWQgUGxhdGZvcm0gTW9kdWxlIEVuZG9yc2VtZW50
MIHXBgNVHSMEgc8wgcyAFCHd7bYICEQX3QxR30ixhppG7bjmoYGwpIGtMIGqMQsw
CQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTESMBAGA1UEBxMJU3Vubnl2
YWxlMRkwFwYDVQQKFBBKdW5pcGVyX05ldHdvcmtzMR0wGwYDVQQLFBRDZXJ0aWZp
Y2F0ZV9Jc3N1YW5jZTEZMBcGA1UEAxQQVFBNX1RydXN0X0FuY2hvcjEdMBsGCSqG
SIb3DQEJARYOY2FAanVuaXBlci5jb22CAQEwcQYDVR0fBGowaDBmoC6gLIYqaHR0
cDovL2NybC5qdW5pcGVyLm5ldD9jYT1KdW5pcGVyX1hYWFhYX0NBojSkMjAwMRMw
EQYDVQQKFApUUE1fVmVuZG9yMRkwFwYDVQQDFBBKdW5pcGVyX1hYWFhYX0NBMFsG
A1UdEQEB/wRRME+kTTBLMQswCQYDVQQGEwJVSzEYMBYGA1UEChMPTXkgT3JnYW5p
emF0aW9uMRAwDgYDVQQLEwdNeSBVbml0MRAwDgYDVQQDEwdNeSBOYW1lMA0GCSqG
SIb3DQEBCwUAA4IBAQCsVFVA90O8E4p/8ohBYQRezVaWidTHCTM1sdAoeljlrsFX
xqwcQEGVT3BpzwN8w2r+iKOKLQkWv64os0KKL0RIIjmCmJ2RukqH/R0M8Air4+Im
iWI3xV+HzVRsJIrCRT2tzxbchU/i/LQiwhteUEZ9sZbHKyLQe9x9HgByM05ifOGh
z2dcb7AWNlo7nJtRBmx0v9iim2kktqGMuXgBzlnMMabqHMb4L+vjww2Wn5nNYbr/
oXq4fa01MGQyvRPAEOwL3ZxcaqKHvmTn9coBLhpP3nQIEV+V+PngQjtBmwdkjIj5
feDp86jGN6348H+z9CzXUSbyOn6utIxN0SvVESxx&lt;/X509Certificate&gt;
        &lt;X509Certificate&gt;\
MIIExTCCA62gAwIBAgIBATANBgkqhkiG9w0BAQsFADCBqjELMAkGA1UEBhMCVVMx
EzARBgNVBAgTCkNhbGlmb3JuaWExEjAQBgNVBAcTCVN1bm55dmFsZTEZMBcGA1UE
ChQQSnVuaXBlcl9OZXR3b3JrczEdMBsGA1UECxQUQ2VydGlmaWNhdGVfSXNzdWFu
Y2UxGTAXBgNVBAMUEFRQTV9UcnVzdF9BbmNob3IxHTAbBgkqhkiG9w0BCQEWDmNh
QGp1bmlwZXIuY29tMB4XDTEzMTAyMDE2MjIwOVoXDTE0MTAyMDE2MjIwOVowMDET
MBEGA1UEChQKVFBNX1ZlbmRvcjEZMBcGA1UEAxQQSnVuaXBlcl9YWFhYWF9DQTCC
ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAK+D34JQ/tsWv5SZ5L2TF7u7
xo7eZEpz/BmnXhxa6keBx5gmjkBXgfSMov7ZJaZfzXkCL01YDDCDQyXBLkh/n2bL
3K0AkEUJPTJgSTTQbPtLkVJgWWAwYASu3/L88c9JH33tvPNQusL0qW683Pd3iVV5
VFOe7c2ZZ0aUtw/FBexjOwPmkQdivb78mfNwyJYkgy0dq0z5GaIIZNna2de1N/Jk
mStZEB6+QJfn0qRsaJbA3TS5JQ13ZBSOqcvtjOIDingjHCXGWEULTeF1UVExNXEG
fsHY2CtQaP/r8hT/8TjPB4mJpbuG1P/BpIAXtBC+hqggwAnNpVfcAxReozzoFCcC
AwEAAaOCAW0wggFpMBIGA1UdEwEB/wQIMAYBAf8CAQAwHQYDVR0OBBYEFCHd7bYI
CEQX3QxR30ixhppG7bjmMIHfBgNVHSMEgdcwgdSAFH+nvIT5PZV62rnjGbqzwT2R
K1FOoYGwpIGtMIGqMQswCQYDVQQGEwJVUzETMBEGA1UECBMKQ2FsaWZvcm5pYTES
MBAGA1UEBxMJU3Vubnl2YWxlMRkwFwYDVQQKFBBKdW5pcGVyX05ldHdvcmtzMR0w
GwYDVQQLFBRDZXJ0aWZpY2F0ZV9Jc3N1YW5jZTEZMBcGA1UEAxQQVFBNX1RydXN0
X0FuY2hvcjEdMBsGCSqGSIb3DQEJARYOY2FAanVuaXBlci5jb22CCQCVivZlfsyT
TzAOBgNVHQ8BAf8EBAMCAgQwQgYDVR0fBDswOTA3oDWgM4YxaHR0cDovL2NybC5q
dW5pcGVyLm5ldD9jYT1KdW5pcGVyX1RydXN0X0FuY2hvcl9DQTANBgkqhkiG9w0B
AQsFAAOCAQEAXw4/3c9yC4TiYTxHmEXoqYgw2+xyEtJIEs3Kv7MSbF/cJwXz4lci
8Fy3ZiKgq9gj9vloWLT5V9ri1HCgalD8D56iKtQCOvY7TJ64qChAA8q7/WNC3dbJ
s9Op6+nSpolfG8YNHfBroCSfNOVCteJ+pU26p3cC1150Pr+/yZZHnsMhNLYuLCvq
29uvnPDBC4MMVfcMbasPpsxL7Ue4PJsjnLquGLZ33MgNGP1TdefvYCFLF2ZEIbvi
KEGLOTXMrXsbUbQLZAdlq6kLCm7A3u6gwTMg+NydCziVsARq+ZKJS0n3vDoAIJxl
BfXhJE4VOjAEQ8w+Sftu1lu6rJZr3ctSLg==&lt;/X509Certificate&gt;
      &lt;/X509Data&gt;
    &lt;/KeyInfo&gt;
```

```
     </Signature>
   </configlet>
```

Appendix B. Change Log

B.1. 00 to 01

Complete re-write. Switched from using signed DNS records using DNSSEC to using signed YANG-defined XML files using XML Signature. This update took into a lot a feedback from both operators and vendors.

Appendix C. Open Issues

C.1. How to best structure the Configlet YANG module?

The current YANG module must redefine parts of the "ietf-system" and "ietf-netconf-server" modules. Also, when referencing parts that it can, the YANG module unnecessarily includes parts it doesn’t need, such as configuring the device to listen for inbound connections. Ideally "deviation" statements could be used to delete the unwanted sub-trees.

C.2. Should Configlets always be signed?

This drafts states that Configlets don’t have to be signed when loaded through a mechanism that asserts physical presence. However, some have voiced concern, saying that no possible backdoor should be allowed.

Authors’ Addresses

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Stephen Hanna
Juniper Networks

EMail: shanna@juniper.net

Joe Marcus Clarke
Cisco Systems

EMail: jclarke@cisco.com

Mikael Abrahamsson
T-Systems

EMail: "mikael.abrahamsson@t-systems.se

Network Working Group T. Mizrahi
Internet Draft Y. Moses
Intended status: Experimental Technion, Israel Institute of Technology
Expires: April 2016 October 15, 2015

Time Capability in NETCONF
draft-mm-netconf-time-capability-09.txt

Abstract

This document defines a capability-based extension to the Network Configuration Protocol (NETCONF) that allows time-triggered configuration and management operations. This extension allows NETCONF clients to invoke configuration updates according to scheduled times, and allows NETCONF servers to attach timestamps to the data they send to NETCONF clients.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

The Network Configuration Protocol (NETCONF) defined in [RFC6241] provides mechanisms to install, manipulate, and delete the configuration of network devices. NETCONF allows clients to configure and monitor NETCONF servers using remote procedure calls (RPC).

NETCONF, as defined in [RFC6241], is asynchronous; when a client invokes an RPC, it has no control over the time at which the RPC is executed, nor does it have any feedback from the server about the execution time.

Time-based configuration ([HotSDN], [TimeTR]) can be a useful tool that enables an entire class of coordinated and scheduled configuration procedures. Time-triggered configuration allows coordinated network updates in multiple devices; a client can invoke a coordinated configuration change by sending RPCs to multiple servers with the same scheduled execution time. A client can also invoke a time-based sequence of updates by sending n RPCs with n different update times, T1, T2, ..., Tn, determining the order in which the RPCs are executed.

This memo defines the :time capability in NETCONF. This extension allows clients to determine the scheduled execution time of RPCs they send. It also allows a server that receives an RPC to report its actual execution time to the client.

The NETCONF time capability is intended for scheduling RPCs that should be performed in the near future, allowing to coordinate simultaneous configuration changes, or to specify an order of configuration updates. Time-of-day-based policies and far-future scheduling, e.g., [Cond], are outside the scope of this memo.

This memo is defined for experimental purposes, and will allow the community to experiment with the NETCONF time capability. It is expected that based on the lessons learned from this experience the NETCONF working group will be able to consider whether to adopt the time capability.

2. Conventions used in this document

2.1. Key words

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 [RFC2119].

2.2. Abbreviations

NETCONF Network Configuration Protocol

RPC Remote Procedure Call

2.3. Terminology

- o Capability [RFC6241]: A functionality that supplements the base NETCONF specification.

- o Client [RFC6241]: Invokes protocol operations on a server. In addition, a client can subscribe to receive notifications from a server.

- o Execution time: The execution time of an RPC is defined as the time at which a server completes the execution of an RPC.

- o Scheduled RPC: an RPC that is scheduled to be performed at a predetermined time, which is included in the <rpc> message.

- o Scheduled time: The scheduled time of an RPC is the time at which the RPC should be invoked. The scheduled time is determined by the client, and enforced by the server.

- o Server [RFC6241]: Executes protocol operations invoked by a client. In addition, a server can send notifications to a client.

3. Using Time in NETCONF

3.1. The Time Capability in a Nutshell

The :time capability provides two main functions:

- o Scheduling:
  When a client sends an RPC to a server, the RPC message MAY include the scheduled-time element, denoted by Ts in Figure 1. The server then executes the RPC at the scheduled time Ts, and once completed the server can respond with an RPC reply message.

- o Reporting:
  When a client sends an RPC to a server, the RPC message MAY include a get-time element (see Figure 2), requesting the server to return the execution time of the RPC. In this case, after the server performs the RPC it responds with an RPC reply that includes the execution time, Te.

```
                      RPC _________
                    executed       \
                                   \/
                                   Ts
            server  ---------------+-------------        ----> time
                              /\      \
                          rpc /        \ rpc-reply
                         (Ts)/          \
                            /           \/
            client  -----------------------------
```

Figure 1 Scheduled RPC

```
                   RPC _________
                 executed       \
                                \/
                                Te
            server  ------------+----------------        ----> time
                              /\   \
                       rpc    /     \ rpc-reply
                   (get-time)/       \ (Te)
                            /        \/
            client  -----------------------------
```

Figure 2 Reporting the Execution Time of an RPC

Example 1. A client needs to trigger a commit at n servers, so that the n servers perform the commit as close as possible to simultaneously. Without the time capability, the client sends a sequence of n commit messages, and thus each server performs the commit at a different time. By using the time capability, the client can send commit messages that are scheduled to take place at a chosen time Ts, for example 5 seconds in the future, causing the servers to invoke the commit as close as possible to time Ts.

Example 2. In many applications it is desirable to monitor events or collect statistics regarding a common time reference. A client can send a set of get-config messages that is scheduled to be executed at multiple servers at the same time, providing a simultaneous system-wide view of the state of the servers. Moreover, a client can use the get-time element in its get-config messages, providing a time reference to the sampled element.

The scenarios of Figure 1 and Figure 2 imply that a third scenario can also be supported (Figure 3), where the client invokes an RPC that includes a scheduled time, Ts, as well as the get-time element. This allows the client to receive feedback about the actual execution time, Te. Ideally, Ts=Te. However, the server may execute the RPC at a slightly different time than Ts, for example if the server is tied up with other tasks at Ts.

```
                  RPC _________
                executed       \
                                \/
                             Ts Te
        server  -------------+-+-------------        ----> time
                            /\        \
               rpc         /           \ rpc-reply
        (Ts + get-time)/              \ (Te)
                         /              \/
        client  -----------------------------
```

Figure 3 Scheduling and Reporting

3.2. Notifications and Cancellation Messages

Notifications

As illustrated in Figure 1, after a scheduled RPC is executed the server sends an rpc-reply. The rpc-reply may arrive a long period of time after the RPC was sent by the client, leaving the client without a clear indication of whether the RPC was received.

This document defines a new notification, the netconf-scheduled-message notification, which provides an immediate acknowledgement of the scheduled RPC.

The <netconf-scheduled-message> is sent to the client if it is subscribed to the NETCONF notifications [RFC6470]; as illustrated in Figure 4, when the server receives a scheduled RPC it sends a notification to the client.

The <netconf-scheduled-message> notification includes a <schedule-id> element. The <schedule-id> is a unique identifier that the server assigns to every scheduled RPC it receives. Thus, a client can keep track of all the pending scheduled RPCs; a client can uniquely identify a scheduled RPC by the tuple {server, schedule-id}.

```
                      RPC ____________
                    executed          \
                                      \/
                                      Ts
            server  -------------------+---------        ----> time
                        /\  \            \
                    rpc /    \notifi-     \ rpc-reply
                   (Ts)/      \cation      \
                      /       \/           \/
            client  -----------------------------
```

Figure 4 Scheduled RPC with Notification

Cancellation Messages

A client can cancel a scheduled RPC by sending a <cancel-schedule> RPC. The <cancel-schedule> RPC includes the <schedule-id> of the scheduled RPC that needs to be cancelled.

The <cancel-schedule> RPC, defined in this document, can be used to perform a coordinated all-or-none procedure, where either all the servers perform the operation on schedule, or the operation is aborted.

Example 3. A client sends scheduled RPC messages to server 1 and server 2, both scheduled to be performed at time Ts. Server 1 sends a notification indicating that it has successfully scheduled the RPC, while server 2 replies with an unknown-element error [RFC6241] that indicates that it does not support the time capability. The client sends a <cancel-schedule> RPC to server 1, and receives an rpc-reply. The message exchange between the client and server 1 in this example is illustrated in Figure 5.

```
                           RPC not __________
                           executed          \
                                             \/
                                              Ts
        server  --------------------------------+---      ----> time
                    /\ \            /\        \
                rpc /   \notifi-    /cancel-   \ rpc-reply
               (Ts)/     \cation   /schedule    \
                  /      \/       /             \/
        client  ------------------------------------
```

Figure 5 Cancellation Message

A cancel-schedule message MUST NOT include the scheduled-time parameter. A server that receives a cancel-schedule should try to cancel the schedule as soon as possible. If the server is unable to cancel the scheduled RPC, for example because it has already been executed, it should respond with an rpc-error [RFC6241], in which the error-type is 'protocol', and the error-tag is 'operation-failed'.

3.3. Synchronization Aspects

The time capability defined in this document requires clients and servers to maintain clocks. It is assumed that clocks are synchronized by a method that is outside the scope of this document, e.g., [NTP] or [IEEE1588].

This document does not define any requirements pertaining to the degree of accuracy of performing scheduled RPCs. Note that two factors affect how accurately the server can perform a scheduled RPC; one factor is the accuracy of the clock synchronization method used to synchronize the clients and servers, and the second factor is the server's ability to execute real-time configuration changes, which greatly depends on how it is implemented. Typical networking devices are implemented by a combination of hardware and software. While the execution time of a hardware module can typically be predicted with a high level of accuracy, the execution time of a software module may be variable and hard to predict. A configuration update would typically require the server's software to be involved, thus affecting how accurately the RPC can be scheduled.

Another important aspect of synchronization, is monitoring; a client should be able to check whether a server is synchronized to a reference time source. Typical synchronization protocols, such as the Network Time Protocol [NTP] provide the means ([RFC5907], [RFC7317]) to verify that a clock is synchronized to a time reference by querying its Management Information Base (MIB). The get-time feature defined in this document (see Figure 2) allows a client to obtain a rough estimate of the time offset between the client's clock and the server's clock.

Since servers do not perform configuration changes instantaneously, the processing time of an RPC should not be overlooked. The scheduled time always refers to the start time of the RPC, and the execution time always refers to its completion time.

3.4. Scheduled Time Format

The scheduled time and execution time fields in RPC messages use a common time format field.

The time format used in this document is the date-and-time format, that is defined in Section 5.6 of [RFC3339] and in Section 3 of [RFC6991].

```
    leaf scheduled-time {
      type yang:date-and-time;
      description
      "The time at which the RPC is scheduled to be performed.";
    }


    leaf execution-time {
      type yang:date-and-time;
      description
      "The time at which the RPC was executed.";
    }
```

3.5. Scheduling Tolerance

When a client sends an RPC that is scheduled to Ts, the server MUST verify that the value Ts is not too far in the past or in the future. As illustrated in Figure 6, the server verifies that Ts is within the scheduling tolerance range.

```
                   RPC _________
                 received       \
                                \/
                                      Ts
             -----+--------------+-----+------------+-------> time

                   <------------> <---------------->
                   sched-max-past  sched-max-future


                   <--------------------------------->
                        scheduling tolerance
```

Figure 6 Scheduling Tolerance

The scheduling tolerance is determined by two parameters, sched-max-future and sched-max-past. These two parameters use the time-interval format (Section 3.7.), and their default value is 15 seconds.

If the scheduled time, Ts is within the scheduling tolerance range, the scheduled RPC is performed; if Ts occurs in the past and within the scheduling tolerance, the server performs the RPC as soon as possible, whereas if Ts is a future time, the server performs the RPC at Ts.

If Ts is not within the scheduling tolerance range, the scheduled RPC is discarded, and the server responds with an error message [RFC6241] with a bad-element error-tag. An example is provided in Section 5.3.

3.6. Near Future Scheduling vs. Far Future Scheduling

The scheduling bound defined by sched-max-future guarantees that every scheduled RPC is restricted to a near future scheduling time.

The scheduling mechanism defined in this document is intended for near future scheduling, on the order of seconds. Far future scheduling is outside the scope of this document.

Example 1 is a typical example of using near future scheduling; the goal in the example is to perform the RPC at multiple servers at the same time, and therefore it is best to schedule the RPC to be performed a few seconds in the future.

The Challenges of Far Future Scheduling

When an RPC is scheduled to be performed at a far-future time, during the long period between the time at which the RPC is sent and the time at which it is scheduled to be executed the following erroneous events may occur:

- o The server may restart.

- o The client's authorization level may be changed.

- o The client may restart and send a conflicting RPC.

- o A different client may send a conflicting RPC.

In these cases if the server performs the scheduled operation it may perform an action that is inconsistent with the current network policy, or inconsistent with the currently active clients.

Near future scheduling guarantees that external events such as the examples above have a low probability of occurring during the sched-max-future period, and even when they do, the period of inconsistency is limited to sched-max-future, which is a short period of time.

The Tradeoff in Setting the sched-max-future Value

The sched-max-future parameter should be configured to a value that is high enough to allow the client to:

1. Send the scheduled RPC, potentially to multiple servers.

2. Receive notifications or rpc-error messages from the server(s), or wait for a timeout and decide that if no response has arrived then something is wrong.

3. If necessary, send a cancellation message, potentially to multiple servers.

On the other hand, sched-max-future should be configured to a value that is low enough to allow a low probability of the erroneous events above, typically on the order of a few seconds. Note that even if sched-max-future is configured to a low value, it is still possible (with a low probability) that an erroneous event will occur. However, this short potentially hazardous period is not significantly worse than in conventional (unscheduled) RPCs, as even a conventional RPC may in some cases be executed a few seconds after it was sent by the client.

The Default Value of sched-max-future

The default value of sched-max-future is defined to be 15 seconds. This duration is long enough to allow the scheduled RPC to be sent by the client, potentially to multiple servers, and in some cases to send a cancellation message, as described in Section 3.2. On the other hand, the 15 second duration yields a very low probability of a reboot or a permission change.

3.7. Time Interval Format

The time-interval format is used for representing the length of a time interval, and is based on the date-and-time format. It is used for representing the scheduling tolerance parameters, as described in the previous section.

While the date-and-time type uniquely represents a specific point in time, the time-interval type defined below can be used to represent the length of a time interval without specifying a specific date.

The time-interval type is defined as follows:

```
typedef time-interval {
  type string {
    pattern '\d{2}:\d{2}:\d{2}(\.\d+)?';
  }
  description
    "Defines a time interval, up to 24 hours.
     The format is specified as HH:mm:ss.f,
     consisting of two digits for hours,
     two digits for minutes, two digits
     for seconds, and zero or more digits
     representing second fractions.";
}
```

Example

The sched-max-future parameter is defined (Appendix A) as a time-interval, as follows:

```
leaf sched-max-future {
  type time-interval;
  default 00:00:15.0;
}
```

The default value specified for sched-max-future is 0 hours, 0 minutes, and 15 seconds.

4. Time Capability

The structure of this section is as defined in Appendix D of [RFC6241].

4.1. Overview

A server that supports the time capability can perform time-triggered operations as defined in this document.

A server implementing the :time capability:

- o MUST support the ability to receive <rpc> messages that include a time element, and perform a time-triggered operation accordingly.
- o MUST support the ability to include a time element in the <rpc-reply> messages that it transmits.

4.2. Dependencies

With-defaults Capability

The time capability YANG module (Appendix A.) uses default values, and thus it is assumed that the with-defaults capability [RFC6243] is supported.

4.3. Capability Identifier

The :time capability is identified by the following capability string (to be assigned by IANA - see Section 0):

urn:ietf:params:netconf:capability:time:1.0

4.4. New Operations

<cancel-schedule>

The cancel-schedule RPC is used for cancelling an RPC that was previously scheduled.

A cancel-schedule RPC MUST include the <cancelled-message-id> element, which specifies the message ID of the scheduled RPC that needs to be cancelled.

A cancel-schedule RPC MAY include the <get-time> element. In this case the rpc-reply includes the <execution-time> element, specifying the time at which the scheduled RPC was cancelled.

4.5. Modifications to Existing Operations

4.5.1. Affected Operations

The :time capability defined in this memo can be applied to any of the following operations:

o get-config

o get

o copy-config

o edit-config

o delete-config

o lock

o unlock

o commit

Three new elements are added to each of these operations:

o <scheduled-time>
   This element is added to the input of each operation, indicating the time at which the server is scheduled to invoke the operation. Every <rpc> message MAY include the <scheduled-time> element. A server that supports the :time capability and receives an <rpc> message with a <scheduled-time> element MUST perform the operation as close as possible to the scheduled time.

   The <scheduled-time> element uses the date-and-time format (Section 3.4.).

o <get-time>
This element is added to the input of each operation. An <rpc> message MAY include a <get-time> element, indicating that the server MUST include an <execution-time> in its corresponding <rpc-reply>.

o <execution-time>
This element is added to the output of each operation, indicating the time at which the server completed the operation. An <rpc-reply> MAY include the <execution-time> element. A server that supports the :time capability and receives an operation with the <get-time> element MUST include the execution time in its response.

The execution-time element uses the date-and-time format (Section 3.4.).

4.5.2. Processing Scheduled Operations

A server that receives a scheduled RPC MUST start executing the RPC as close as possible to its scheduled execution time.

If a session between a client and a server is terminated, the server MUST cancel all pending scheduled RPCs that were received in this session.

Scheduled RPCs are processed serially, in an order that is defined by their scheduled times. Thus, the server sends <rpc-reply> messages to scheduled RPCs according to the order of their corresponding schedules. Note that this is a modification to the behavior defined in [RFC6241], which states that replies are sent in the order the requests were received. Interoperability with [RFC6241] is guaranteed by the NETCONF capability exchange; a server that does not support the :time capability responds to RPCs in the order the requestes were received. A server that supports the :time capability replies to conventional (non-scheduled) RPCs in the order they were received, and replies to scheduled RPCs in the order of their scheduled times.

If a server receives two or more RPCs that are scheduled to be performed at the same time, the server executes the RPCs serially in an arbitrary order.

4.6. Interactions with Other Capabilities

Confirmed Commit Capability

The confirmed commit capability is defined in Section 8.4 of [RFC6241]. According to [RFC6241], a confirmed <commit> operation MUST be reverted if a confirming commit is not issued within the timeout period (which by default is 600 seconds).

When the time capability is supported, and a confirmed <commit> operation is used with the <scheduled-time> element, the confirmation timeout MUST be counted from the scheduled time, i.e., the client begins the timeout measurement starting at the scheduled time.

5. Examples

5.1. <scheduled-time> Example

The following example extends the example presented in Section 7.2 of [RFC6241] by adding the time capability. In this example, the <scheduled-time> element is used to specify the scheduled execution time of the configuration update (as shown in Figure 1).

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <scheduled-time
       xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-time">
        2015-10-21T04:29:00.235Z
    </scheduled-time>
    <config>
      <top xmlns="http://example.com/schema/1.2/config">
        <interface>
          <name>Ethernet0/0</name>
          <mtu>1500</mtu>
        </interface>
      </top>
    </config>
  </edit-config>
</rpc>

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

```
</rpc-reply>
```

5.2. <get-time> Example

The following example is similar to the one presented in Section 5.1. , except that in this example the client includes a <get-time> element in its RPC, and the server consequently responds with an <execution-time> element (as shown in Figure 2).

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <get-time
     xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-time">
    </get-time>
    <config>
      <top xmlns="http://example.com/schema/1.2/config">
        <interface>
          <name>Ethernet0/0</name>
          <mtu>1500</mtu>
        </interface>
      </top>
    </config>
  </edit-config>
</rpc>

<rpc-reply message-id="101"
     xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <ok/>
  <execution-time>
      2015-10-21T04:29:00.235Z
  </execution-time>
</rpc-reply>
```

5.3. Error Example

The following example presents a scenario in which the scheduled-time is not within the scheduling tolerance, i.e., it is too far in the past, and therefore an rpc-error is returned.

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <edit-config>
    <target>
      <running/>
    </target>
    <scheduled-time
       xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-time">
        2010-10-21T04:29:00.235Z
    </scheduled-time>
    <config>
      <top xmlns="http://example.com/schema/1.2/config">
        <interface>
          <name>Ethernet0/0</name>
          <mtu>1500</mtu>
        </interface>
      </top>
    </config>
  </edit-config>
</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>bad-element</error-tag>
    <error-severity>error</error-severity>
    <error-info>
      <bad-element>scheduled-time</bad-element>
    </error-info>
  </rpc-error>
</rpc-reply>
```

6. Security Considerations

6.1. General Security Considerations

The security considerations of the NETCONF protocol in general are discussed in [RFC6241].

The usage of the time capability defined in this document can assist an attacker in gathering information about the system, such as the exact time of future configuration changes. Moreover, the time elements can potentially allow an attacker to learn information about the system’s performance. Furthermore, an attacker that sends malicious RPC messages can use the time capability to amplify her attack; for example, by sending multiple RPC messages with the same scheduled time. It is important to note that the security measures described in [RFC6241] can prevent these vulnerabilities.

The time capability relies on an underlying time synchronization protocol. Thus, by attacking the time protocol an attack can potentially compromise NETCONF when using the time capability. A detailed discussion about the threats against time protocols and how to mitigate them is presented in [TimeSec].

The time capability can allow an attacker to attack a NETCONF server by sending malicious RPCs that are scheduled to take place in the future. For example, an attacker can send multiple scheduled RPCs that are scheduled to be performed at the same time. Another possible attack is to send a large number of scheduled RPCs to a NETCONF server, potentially causing the server’s buffers to overflow. These attacks can be mitigated by a carefully designed NETCONF server; when a server receives a scheduled RPC that exceeds its currently available resources, it should reply with an rpc-error, and discard the scheduled RPC.

Note that if an attacker has been detected and revoked, its future scheduled RPCs are not executed; as defined in Section 4.5.2. , once the session with the attacker has been terminated, the corresponding scheduled RPCs are discarded.

6.2. YANG Module Security Considerations

This memo defines a new YANG module, as specified in Appendix A.

The YANG module defined in this memo is designed to be accessed via the NETCONF protocol [RFC6241]. The lowest NETCONF layer is the secure transport layer and the mandatory to implement secure transport is SSH [RFC6242]. The NETCONF access control model [RFC6536] provides the means to restrict access for particular NETCONF users to a pre-configured subset of all available NETCONF protocol operations and content.

This YANG module defines <sched-max-future> and <sched-max-past>, which are writable/creatable/deletable. These data nodes may be considered sensitive or vulnerable in some network environments. An attacker may attempt to maliciously configure these parameters to a low value, thereby causing all scheduled RPCs to be discarded. For instance, if a client expects <sched-max-future> to be 15 seconds, but in practice it is maliciously configured to 1 second, then a legitimate scheduled RPC that is scheduled to be performed 5 seconds in the future will be discarded by the server.

This YANG module defines the <cancel-schedule> RPC. This RPC may be considered sensitive or vulnerable in some network environments. Since the value of the <schedule-id> is known to all the clients that are subscribed to notifications from the server, the <cancel-schedule> RPC may be used maliciously to attack servers by canceling their pending RPCs. This attack is addressed in two layers: (i) security at the transport layer, limiting the attack only to clients that have successfully initiated a secure session with the server, and (ii) the authorization level required to cancel an RPC should be the same as the level required to schedule it, limiting the attack only to attackers with an authorization level that is equal to or higher than that of the client that initiated the scheduled RPC.

7. IANA Considerations

This document proposes to register the following capability identifier URN in the 'Network Configuration Protocol (NETCONF) Capability URNs' registry:

urn:ietf:params:netconf:capability:time:1.0

This document proposes to register the following XML namespace URN in the 'IETF XML registry', following the format defined in [RFC3688]:

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-time

This document proposes to register a module name in the 'YANG Module Names' registry, defined in [RFC6020].

name: ietf-netconf-time

prefix: nct

namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-time

RFC: TBD

8. Acknowledgments

The authors gratefully acknowledge Joe Marcus Clarke, Andy Bierman, Balazs Lengyel, Jonathan Hansford, John Heasley, Robert Sparks, Al

Morton, Olafur Gudmundsson, Juergen Schoenwaelder, Joel Jaeggli, Alon Schneider and Eylon Egozi for their insightful comments.

This work was supported in part by Israel Science Foundation grant ISF 1520/11.

This document was prepared using 2-Word-v2.0.template.dot.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3339] Klyne, G., Ed. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, January 2004.

[RFC6991] Schoenwaelder, J., "Common YANG Data Types", RFC 6991, July 2013.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., Bierman, A., Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC6470] Bierman, A., "Network Configuration Protocol (NETCONF) Base Notifications", RFC 6470, February 2012.

9.2. Informative References

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.

[RFC6243] Bierman, A., Lengyel, B., "With-defaults Capability for NETCONF", RFC 6243, June 2011.

[RFC6536] Bierman, A. and M. Bjorklund, "Network Configuration Protocol (NETCONF) Access Control Model", RFC 6536, March 2012.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, August 2014.

[Cond] Watsen, K., "Conditional Enablement of Configuration Nodes", draft-kwatsen-conditional-enablement-00 (expired), 2013.

[HotSDN] Mizrahi, T., Moses, Y., "Time-based Updates in Software Defined Networks", the second workshop on hot topics in software defined networks (HotSDN), 2013.

[IEEE1588] IEEE TC 9 Instrumentation and Measurement Society, "1588 IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems Version 2", IEEE Standard, 2008.

[NTP] Mills, D., Martin, J., Burbank, J., Kasch, W., "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

[RFC5907] Gerstung, H., Elliott, C., Haberman, B., "Definitions of Managed Objects for Network Time Protocol Version 4 (NTPv4", RFC 5907, June 2010.

[TimeSec] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, October 2014.

[TimeTR] Mizrahi, T., Moses, Y., "Time-based Updates in OpenFlow: A Proposed Extension to the OpenFlow Protocol", Technion - Israel Institute of Technology, technical report, CCIT Report #835, EE Pub No. 1792, 2013.
http://tx.technion.ac.il/˜dew/OFTimeTR.pdf

Appendix A. YANG Module for the Time Capability

This section is normative.

```
<CODE BEGINS> file "ietf-netconf-time@2015-09-01.yang"

module ietf-netconf-time {

   namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-time";

   prefix nct;
```

```
   import ietf-netconf { prefix nc; }

   import ietf-yang-types { prefix yang; }

   import ietf-netconf-monitoring { prefix ncm; }

   organization
     "IETF";

   contact
     "Editor: Tal Mizrahi
         <dew@tx.technion.ac.il>
      Editor: Yoram Moses
         <moses@ee.technion.ac.il>";

   description
     "This module defines a capability-based extension to the
      Network Configuration Protocol (NETCONF) that allows
      time-triggered configuration and management operations.
      This extension allows NETCONF clients to invoke configuration
      updates according to scheduled times, and allows NETCONF
      servers to attach timestamps to the data they send to NETCONF
      clients.

      Copyright (c) 2015 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).";

   revision 2015-09-01 {
     description
       "Initial version.";
     reference
       "draft-mm-netconf-time-capability:
        Time Capability in NETCONF";
   }



   typedef time-interval {
     type string {
```

```
           pattern '\d{2}:\d{2}:\d{2}(\.\d+)?';
         }
         description
           "Defines a time interval, up to 24 hours.
            The format is specified as HH:mm:ss.f,
            consisting of two digits for hours,
            two digits for minutes, two digits
            for seconds, and zero or more digits
            representing second fractions.";
       }


       grouping scheduling-tolerance-parameters {
         leaf sched-max-future {
           type time-interval;
           default 00:00:15.0;
           description
             "When the scheduled time is in the future, i.e., greater
              than the present time, this leaf defines the maximal
              difference between the scheduled time
              and the present time that the server is willing to
              accept. If the difference exceeds this number, the
              server responds with an error.";
         }

         leaf sched-max-past {
           type time-interval;
           default 00:00:15.0;
           description
             "When the scheduled time is in the past, i.e., less
              than the present time, this leaf defines the maximal
              difference between the present time
              and the scheduled time that the server is willing to
              accept. If the difference exceeds this number, the
              server responds with an error.";
         }

         description
           "Contains the parameters of the scheduling tolerance.";
       }
```

```
      // extending the get-config operation
      augment /nc:get-config/nc:input {
        leaf scheduled-time {
          type yang:date-and-time;
          description
            "The time at which the RPC is scheduled to be performed.";
        }

        leaf get-time {
          type empty;
          description
            "Indicates that the rpc-reply should include the
             execution-time.";
        }

        description
          "Adds the time element to <get-config>.";
      }

      augment /nc:get-config/nc:output {
        leaf execution-time {
          type yang:date-and-time;
          description
            "The time at which the RPC was executed.";
        }

        description
          "Adds the time element to <get-config>.";
      }


      augment /nc:get/nc:input {
        leaf scheduled-time {
          type yang:date-and-time;
          description
            "The time at which the RPC is scheduled to be performed.";
        }

        leaf get-time {
          type empty;
```

```
          description
            "Indicates that the rpc-reply should include the
             execution-time.";
        }

        description
          "Adds the time element to <get>.";
      }

      augment /nc:get/nc:output {
        leaf execution-time {
          type yang:date-and-time;
          description
            "The time at which the RPC was executed.";
        }

        description
          "Adds the time element to <get>.";
      }

      augment /nc:copy-config/nc:input {
        leaf scheduled-time {
          type yang:date-and-time;
          description
            "The time at which the RPC is scheduled to be performed.";
        }

        leaf get-time {
          type empty;
          description
            "Indicates that the rpc-reply should include the
             execution-time.";
        }

        description
          "Adds the time element to <copy-config>.";
      }

      augment /nc:copy-config/nc:output {
        leaf execution-time {
          type yang:date-and-time;
```

```
          description
            "The time at which the RPC was executed.";
        }

        description
          "Adds the time element to <copy-config>.";
      }

      augment /nc:edit-config/nc:input {
        leaf scheduled-time {
          type yang:date-and-time;
          description
            "The time at which the RPC is scheduled to be performed.";
        }

        leaf get-time {
          type empty;
          description
            "Indicates that the rpc-reply should include the
             execution-time.";
        }

        description
          "Adds the time element to <edit-config>.";
      }

      augment /nc:edit-config/nc:output {
        leaf execution-time {
          type yang:date-and-time;
          description
            "The time at which the RPC was executed.";
        }

        description
          "Adds the time element to <edit-config>.";
      }

      augment /nc:delete-config/nc:input {
        leaf scheduled-time {
          type yang:date-and-time;
          description
```

```
            "The time at which the RPC is scheduled to be performed.";
        }

        leaf get-time {
          type empty;
          description
            "Indicates that the rpc-reply should include the
             execution-time.";
        }

        description
         "Adds the time element to <delete-config>.";
      }

      augment /nc:delete-config/nc:output {
        leaf execution-time {
          type yang:date-and-time;
          description
            "The time at which the RPC was executed.";
        }
        description
          "Adds the time element to <delete-config>.";
      }

      augment /nc:lock/nc:input {
        leaf scheduled-time {
          type yang:date-and-time;
          description
            "The time at which the RPC is scheduled to be performed.";
        }

        leaf get-time {
          type empty;
          description
            "Indicates that the rpc-reply should include the
             execution-time.";
        }

        description
          "Adds the time element to <lock>.";
      }
```

```
   augment /nc:lock/nc:output {
     leaf execution-time {
       type yang:date-and-time;
       description
         "The time at which the RPC was executed.";
     }

     description
       "Adds the time element to <lock>.";
   }

   augment /nc:unlock/nc:input {
     leaf scheduled-time {
       type yang:date-and-time;
       description
         "The time at which the RPC is scheduled to be performed.";
     }

     leaf get-time {
       type empty;
       description
         "Indicates that the rpc-reply should include the
          execution-time.";
     }

     description
       "Adds the time element to <unlock>.";
   }

   augment /nc:unlock/nc:output {
     leaf execution-time {
       type yang:date-and-time;
       description
         "The time at which the RPC was executed.";
     }

     description
       "Adds the time element to <unlock>.";
   }
```

```
      augment /nc:commit/nc:input {
        leaf scheduled-time {
          type yang:date-and-time;
          description
            "The time at which the RPC is scheduled to be performed.";
        }

        leaf get-time {
          type empty;
          description
            "Indicates that the rpc-reply should include the
             execution-time.";
        }

        description
          "Adds the time element to <commit>.";
      }

      augment /nc:commit/nc:output {
        leaf execution-time {
          type yang:date-and-time;
          description
            "The time at which the RPC was executed.";
        }

        description
          "Adds the time element to <commit>.";
      }

      augment /ncm:netconf-state {
        container scheduling-tolerance {
          uses scheduling-tolerance-parameters;
          description
            "The scheduling tolerance when the time capability
             is enabled.";
        }
        description
          "The scheduling tolerance of the server.";
      }

      rpc cancel-schedule {
```

```
        description
          "Cancels a scheduled message.";
        reference
          "draft-mm-netconf-time-capability:
           Time Capability in NETCONF";

        input {
          leaf cancelled-message-id {
            type string;
            description
              "The ID of the message to be cancelled.";
          }
          leaf get-time {
            type empty;
            description
              "Indicates that the rpc-reply should include
               the execution-time.";
          }
        }
        output {
          leaf execution-time {
            type yang:date-and-time;
            description
              "The time at which the RPC was executed.";
          }
        }
      }

      notification netconf-scheduled-message {
        leaf schedule-id {
          type string;
          description
            "The ID of the scheduled message.";
        }

        leaf scheduled-time {
          type yang:date-and-time;
          description
            "The time at which the RPC is scheduled to be performed.";
        }
```

```
        description
          "Indicates that a scheduled message was received.";
        reference
          "draft-mm-netconf-time-capability:
           Time Capability in NETCONF";
      }

   }
   <CODE ENDS>
```

Authors' Addresses

Tal Mizrahi
Department of Electrical Engineering
Technion - Israel Institute of Technology
Technion City, Haifa, 32000, Israel

Email: dew@tx.technion.ac.il

Yoram Moses
Department of Electrical Engineering
Technion - Israel Institute of Technology
Technion City, Haifa, 32000, Israel

Email: moses@ee.technion.ac.il

NETCONF Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 02, 2014
R. Varga
Pantheon Technologies SRO
March 3, 2014

Efficient XML Interchange Capability for NETCONF
draft-varga-netconf-exi-capability-02

Abstract

The Network Configuration Protocol (NETCONF) provides mechanisms to install, manipulate, and delete the configuration of network devices via exchange of XML messages in textual representation. Efficient XML Interchange (EXI) is a W3C-recommended binary representation of XML Information Set, which is more efficient from both CPU and bandwidth utilization perspective. This document defines a capability-based extension to the NETCONF protocol that allows peers to agree to exchange protocol messages using EXI encoding.

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 [RFC2119].

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The NETCONF protocol [RFC6241] is defined in terms of two peers, client and server, exchanging XML messages in an RPC pattern. These messages are encoded as XML text documents, which makes the exchange easily understandable by human operators by simply observing them on the wire. Unfortunately, this feature takes its toll on both computation and network resources, as the representation contains redundant information and verbose names.

Efficient XML Interchange [W3C.REC-exi-20110310] is a W3C Recommendation which defines a more efficient way of encoding XML Information Set than the usual text representation. This is achieved by a combination of reduced verbosity, binary encoding and, optionally, pruning of non-essential information like comments.

It seems advantageous to allow clients and servers participating on a NETCONF session to sacrifice human readability to increase processing efficiency, especially in environments with high transactional activity and/or limited computing resources.

2. Terminology

This document uses the following terms defined in [RFC6241]:

- capability
- client
- message
- protocol operation
- remote procedure call
- server

3. EXI Capability

3.1. Overview

The :exi capability indicates that the peer supports EXI message encoding and is willing to use it. The capability has no effect on data being handled by the NETCONF protocol, nor does it affect protocol message exchanges.

3.2. Dependencies

EXI-encoded documents are binary data, this capability may only be used when the underlying transport is 8-bit clean and preserves message boundaries in face of arbitrary binary data. Notably this requires use of Chunked Framing mechanism as described in [RFC6242] when used with SSH transport.

The optional Dynamic Schema-informed Encoding Negotiation mechanism relies on NETCONF Monitoring as defined in [RFC6022].

3.3. Capability Identifier

The EXI capability is identified by the following capability string:

urn:ietf:params:netconf:capability:exi:1.0

The identifier MAY have a the following parameters:

compression: This indicates that the sender is willing to perform EXI compression. The parameter MUST contain a positive integral value, which indicates maximum compression block size which the sender can process.

schemas: This indicates that the sender can use schema-informed grammars for EXI encoding. The parameter MUST contain a value, which has to be one of "builtin", "base:1.1" or "dynamic".

builtin Indicates the ability to use the XML schema built into the EXI specification.

base:1.1 Superset of "builtin", indicates that the sender additionally supports schema-informed EXI encoding, based on netconf.xsd schema published in [RFC6241].

dynamic Superset of "base:1.1", indicates that the sender additionally supports dynamic schema-informed encoding negotiation outlined below.

Examples:

urn:ietf:params:netconf:capability:exi:1.0?compression=1000000

urn:ietf:params:netconf:capability:exi:1.0?schemas=builtin

urn:ietf:params:netconf:capability:exi:1.0?schemas=base:1.1

urn:ietf:params:netconf:capability:exi:1.0?compression=20000&schem
as=builtin

urn:ietf:params:netconf:capability:exi:1.0?schemas=dynamic

3.4. Dynamic Schema-informed Encoding Negotiation

The core of this extension relies on shared knowledge between the server and the client where schema-informed encoding is concerned. This limits the encoding efficiency as the actual data transferred over the session is encoded using the equivalent of the builtin schema. Alleviating this limitation requires a mechanism for discovering data schemas and a protocol for synchronizing their activation.

The base schema discovery mechanism is already present in [RFC6022]. This document extends the /netconf-state/schemas/schema subtree with a new leaf, exi-useable, which indicates whether the server supports the use of that particular schema in the EXI schema-informed encoding process.

The negotiation of use of a particular schema for encoding has multiple aspects. First and foremost is the concern of constrained environments, which may have limited resources and thus their ability to dedicate them to improving encoding efficiency may change over lifetime of a NETCONF session. The second issue comes from the need to synchronize the values used in the "schema" EXI header field. Both end of the session need to map names to the same schemas, otherwise the decoding process will not succeed. This name is carried verbatim in the stream, so it should be as concise as possible.

When the peers have both indicated support for Dynamic Schema-informed Encoding, encoding starts in base:1.1 mode. The client then queries the server for the list of schemas, looking for schemas which have the exi-useable leaf set to true. It then selects the schemas it can use in EXI encoding process, potentially requesting them from the server. Finally it prioritizes them and sends a <enable-schema-encoding> request for each of them. Once the server has assigned a EXI schema-id and communicated it back the the client, both parties can use this schema in EXI encoding. The client can request the end of use of a particular schema via the <disable-schema-encoding> RPC, which the server SHOULD NOT fail.

3.5. New Mandatory Operations

3.5.1. <start-exi>

Description: The <start-exi> operation requests that the message encoding be switched to EXI. The operation is invoked by the client and validated by the server. If the server finds the parameters acceptable, it will issue a positive response in the current session encoding. It MUST encode all subsequent messages using EXI encoding with the supplied parameters. It will also expect all incoming messages to be EXI-encoded. The client MUST NOT send any messages to the server between the time is sends this request and the time it receives a response. Once it receives a positive reply, it MUST encode all subsequent messages using the EXI encoding with the parameters supplied in the RPC. If the operation fails, the session message encoding remains unchanged.

Parameters:

alignment: Requested EXI alignment. If this parameter is not present, bit-packed is assumed. The following values are valid:

bit-packed: Set EXI alignment to bit-packed.

byte-aligned: Set EXI alignment to byte-aligned.

pre-compression: Set EXI alignment to pre-compression.

compressed: Do not specify EXI alignment, but perform EXI compression instead.

fidelity: Requested EXI fidelity options. If this parameter is not present or empty, all fidelity options are disabled. The following items may be specified:

<comments/>: Preserve.comments EXI Fidelity option

<dtd/>: Preserve.dtd EXI Fidelity option

<lexical-values/>: Preserve.lexicalValues EXI Fidelity option

<pis/>: Preserve.pis EXI Fidelity option

<prefixes/>: Preserve.prefixes EXI Fidelity option

schema: Optional parameter. This specifies what schema options should be enabled in the EXI encoding process. The following values are valid:

none Do not use schema-informed grammars at all. This translates to using schemaId of <xsd:nil>true</xsd:nil> in the EXI Options header.

builtin Do no use schema-informed grammars, but use the built-in XML data types. This translates to using an empty schemaId in the EXI Options header.

base:1.1 Use schema-informed grammar based on netconf.xsd as published in [RFC6241] in non-strict mode. The value "base:1.1" should be carried in the schemaId field in the EXI Options.

dynamic Same as base:1.1 with the additional support for dynamically modifying which schemas are available for schema-informed encoding.

Positive Response: If the device was able to satisfy the request, an <rpc-reply> is sent that contains an <ok> element.

Negative Response: An <rpc-error> element is included in the <rpc-reply> if the request cannot be completed for any reason.

Example:

```
<rpc message-id="101"
    xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
    <start-exi>
        <alignment>pre-compression</alignment>
        <fidelity>
            <dtd/>
            <lexical-values/>
        </fidelity>
    </start-exi>
```

```
</rpc>

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

3.5.2. <stop-exi>

Description: The <stop-exi> operation requests that the message encoding be switched to textual XML. The operation is invoked by the client and validated by the server. If the server is able to switch the encoding to XML, it will issue a positive response in the current session encoding. It MUST encode all subsequent messages using standard XML encoding. It will also expect all incoming messages to be XML-encoded. The client MUST NOT send any messages to the server between the time is sends this request and the time it receives a response. Once it receives a positive reply, it MUST encode all subsequent messages using the standard XML encoding. If the operation fails, the session message encoding remains unchanged. If the session currently uses XML encoding, this RPC is a no-operation and SHOULD NOT fail.

Positive Response: If the device was able to satisfy the request, an <rpc-reply> is sent that contains an <ok> element.

Negative Response: An <rpc-error> element is included in the <rpc-reply> if the request cannot be completed for any reason.

Example:

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

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

3.6. New Optional Operations

3.6.1. <enable-schema-encoding>

Description: The <enable-schema-encoding> requests the device assign a numeric identifier for use of a specific schema for EXI Schema-informed encoding.

Parameters:

identifier: Schema identifier, as defined in [RFC6022].

version: Schema version, as defined in [RFC6022].

format: Schema format, as defined in [RFC6022].

Positive Response: If the device was able to satisfy the request, an <rpc-reply> is sent that contains an <exi-schema-id> element, which contains the numeric identifier which should be used in the schemaId EXI header field. This identifier has to be unique.

Negative Response: An <rpc-error> element is included in the <rpc-reply> if the request cannot be completed for any reason.

3.6.2. <disable-schema-encoding>

Description: The <disable-schema-encoding> requests the device to deallocate the schema ID from use on this session and stop using it for encoding data towards the client.

Parameters:

exi-schema-id: EXI Schema ID, as assigned by a previous <enable-schema-encoding> call.

Positive Response: If the device was able to satisfy the request, an <rpc-reply> is sent that contains an <ok> element.

Negative Response: An <rpc-error> element is included in the <rpc-reply> if the request cannot be completed for any reason.

4. YANG module for <start-exi> and <stop-exi> Operations

The following YANG module defines the new operations introduced in this document. The YANG language is defined in [RFC6020]. Every NETCONF speaker that supports the :exi capability MUST implement this YANG module.

```
<CODE BEGINS> file "ietf-netconf-exi@2014-03-03.yang"

module ietf-netconf-exi {
    // vi: set et smarttab sw=4 tabstop=4:
    namespace "urn:ietf:params:xml:ns:netconf:exi:1.0";

    prefix exi;

    import ietf-netconf-monitoring {
        prefix ncm;
        revision-date "2010-10-04";
    }

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

    contact
        "Robert Varga <robert.varga@pantheon.sk>";

    description
        "NETCONF Protocol Operations for Efficient XML Interchange.";

    revision 2014-03-03 {
        description
            "Updated with dynamic schema negotiation.";
        reference
            "I-D.varga-netconf-exi-capability-02";
    }

    revision 2013-10-21 {
        description
            "Initial revision";
        reference
            "I-D.varga-netconf-exi-capability-01";
    }

    typedef exi-alignment {
        type enumeration {
            enum bit-packed {
                description
                    "Use bit-packed EXI alignment";
            }
            enum byte-aligned {
                description
                    "Use byte-aligned EXI alignment";
            }
            enum pre-compression {
                description
                    "Use pre-compression EXI alignment";
            }
            enum compressed {
                description
```

```
                        "Do not set EXI alignment, use EXI compression
                        instead";
                }
            }

            description "EXI alignment specification.";
        }

        typedef exi-fidelity {
            type enumeration {
                enum comments {
                    description
                        "Preserve.comments EXI Fidelity option";
                }
                enum dtd {
                    description
                        "Preserve.dtd EXI Fidelity option";
                }
                enum lexical-values {
                    description
                        "Preserve.lexicalValues EXI Fidelity option";
                }
                enum pis {
                    description
                        "Preserve.pis EXI Fidelity option";
                }
                enum prefixes {
                    description
                        "Preserve.prefixes EXI Fidelity option";
                }
            }

            description "EXI fidelity options.";
        }

        rpc start-exi {
            description
                "Start encoding protocol messages in Efficient XML
                Interchange format.";

            reference "I-D.varga-netconf-exi-capability";

            input {
                leaf alignment {
                    type exi-alignment;
                    default "bit-packed";
                    description "EXI alignment to use.";
                }

                leaf-list fidelity {
                    type exi-fidelity;
                    description "EXI fidelity options to use.";
                }
            }
```

```
       }

       rpc stop-exi {
           description
               "Stop encoding protocol messages in Efficient XML
               Interchange format. Revert back to using the usual text
               XML encoding.";
       }

       grouping schema-identifier {
           description
               "The globally-unique identifier of a schema. This
               grouping contains the verbatim transcription of arguments
               to <get-schema> RPC as defined in RFC6022, except all
               leaves are made mandatory.";

           leaf identifier {
               type string;
               mandatory true;
               description
                   "Identifier for the schema list entry.";
           }

           leaf version {
               type string;
               description
                   "Version of the schema requested.  If this parameter
                   is not present, and more than one version of the
                   schema exists on the server, a 'data-not-unique'
                   error is returned, as described above.";
           }

           leaf format {
               type identityref {
                   base ncm:schema-format;
               }
               description
                   "The data modeling language of the schema.  If this
                   parameter is not present, and more than one formats
                   of the schema exists on the server, a
                   'data-not-unique' error is returned, as described
                   above.";
           }
       }

       typedef exi-schema-id {
           type uint16;
           description
               "Schema identifier for use in the EXI stream header.";
       }

       augment "/ncm:netconf-state/ncm:schemas/ncm:schema" {
           description
               "Additional information about schemas useful for EXI
```

```
                encoding";

            leaf exi-useable {
                type boolean;
                default false;
                description
                    "Set to true if the device can use the schema for EXI
                    Schema-informed encoding.";
            }
            leaf exi-schema-id {
                type exi-schema-id;
                description
                    "The EXI schema ID currently assigned to this schema.
                    This value has meaning only within the session and
                    may differ on other sessions.";
            }
        }

        rpc enable-schema-encoding {
            description
                "Request the use of specificied schema in EXI message
                encoding.  This request is sent by the client to the
                server. If the server is able to transition into using
                the schema, it assigns it a unique EXI integer
                identifier.  This identifier is to be used in the EXI
                header as schema identifier.

                The server may start using the identifier as soon as it
                enqueus the response. The client may start using the
                identifier as soon as it sees this RPC complete.";
            input {
                uses schema-identifier;
            }
            output {
                leaf exi-schema-id {
                    type exi-schema-id;
                    mandatory true;
                    description
                        "The EXI Schema ID assigned to this schema for
                        encoding purposes.";
                }
            }
        }

        rpc disable-schema-encoding {
            description
                "This RPC is send by the client when it stops using a
                particular exi-schema-id.";
            input {
                leaf exi-schema-id {
                    type exi-schema-id;
                    mandatory true;
                    description
                        "The EXI Schema ID which should be disabled.";
```

```
            }
        }
    }
}
```

5. IANA considerations

This document registers the following capability identifier URN in the 'Network Configuration Protocol (NETCONF) Capability URNs' registry: urn:ietf:params:netconf:capability:exi:1.0

6. Security Considerations

The compression option present in EXI specification may increase CPU and memory requirements for encoding the response. Devices should ensure this overhead is acceptable before agreeing to using EXI encoding, such that no operational risks are introduced.

7. Acknowledgements

The author would like to thank Anton Tkacik, Miroslav Miklus and Stefan Kobza for their contributions to this document.

8. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.

[RFC6022] Scott, M. and M. Bjorklund, "YANG Module for NETCONF Monitoring", RFC 6022, October 2010.

[RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.

[W3C.REC-exi-20110310] Schneider, J. and T. Kamiya, "Efficient XML Interchange (EXI) Format 1.0", World Wide Web Consortium Recommendation REC-exi-20110310, March 2011, <http://www.w3.org/TR/2011/REC-exi-20110310>.

Author's Address

Robert Varga
Pantheon Technologies SRO
Mlynske Nivy 56
Bratislava 821 05
Slovakia

Email: robert.varga@pantheon.sk