

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 3, 2013

A. Bierman
YumaWorks
M. Bjorklund
Tail-f Systems
November 30, 2012

YANG-API Protocol
draft-bierman-netconf-yang-api-01

Abstract

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

Status of this Memo

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

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

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

This Internet-Draft will expire on June 3, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	4
1.1.	Simple Subset of NETCONF Functionality	4
1.2.	Data Model Driven API	5
1.3.	Terminology	6
1.3.1.	NETCONF	6
1.3.2.	HTTP	7
1.3.3.	YANG	7
1.3.4.	Terms	8
1.4.	Overview	8
1.4.1.	Resource URI Map	9
1.4.2.	YANG-API Message Examples	9
2.	Framework	15
2.1.	Message Model	15
2.2.	Resource Model	15
2.2.1.	YANG-API Resource Types	15
2.2.2.	Resource Discovery	16
2.3.	Datastore Model	16
2.3.1.	Content Model	17
2.3.2.	Editing Model	17
2.3.3.	Locking Model	19
2.3.4.	Persistence Model	19
2.3.5.	Defaults Model	19
2.4.	Transaction Model	20
2.5.	Extensibility Model	20
2.6.	Versioning Model	20
2.7.	Retrieval Filtering Model	21
2.8.	Access Control Model	21
3.	Operations	22
3.1.	OPTIONS	22
3.2.	HEAD	23
3.3.	GET	24
3.4.	POST	26
3.5.	PUT	26
3.6.	PATCH	27
3.7.	DELETE	27
3.8.	Query Parameters	28
3.8.1.	"config" Parameter	28
3.8.2.	"depth" Parameter	29
3.8.3.	"format" Parameter	30
3.8.4.	"insert" Parameter	30
3.8.5.	"point" Parameter	31
3.8.6.	"select" Parameter	32
3.9.	Protocol Operations	32
4.	Messages	34
4.1.	Request URI Structure	34
4.2.	Message Headers	35

4.3.	Message Encoding	36
4.4.	Return Status	36
4.5.	Message Caching	37
5.	Resources	38
5.1.	API Resource (/yang-api)	38
5.1.1.	/yang-api/datastore	38
5.1.2.	/yang-api/modules	38
5.1.3.	/yang-api/operations	38
5.1.4.	/yang-api/version	40
5.2.	Datastore Resource	41
5.3.	Data Resource	41
5.3.1.	Encoding YANG Instance Identifiers in the Request URI	42
5.3.2.	Identifying YANG-defined Data Resources	44
5.3.3.	Identifying Optional Keys	45
5.3.4.	Data Resource Retrieval	45
5.4.	Operation Resource	47
5.4.1.	Encoding Operation Input Parameters	48
5.4.2.	Encoding Operation Output Parameters	49
5.4.3.	Identifying YANG-defined Operation Resources	50
6.	Error Reporting	51
6.1.	Error Response Message	52
7.	RelaxNG Grammar	55
8.	YANG-API module	56
9.	IANA Considerations	58
10.	Security Considerations	59
11.	Change Log	60
11.1.	00-01	60
12.	Open Issues	61
13.	Example YANG Module	63
14.	Normative References	67
	Authors' Addresses	68

1. Introduction

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

This document describes a RESTful protocol called YANG-API, 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 and operational data. RESTful operations are used to access the hierarchical data within a datastore.

A RESTful 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 RESTful design principles. Since NETCONF protocol operations are not relevant, the user should not need any prior knowledge of NETCONF in order to use the RESTful 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-model specific protocol operations defined with the YANG "rpc" statement can be invoked with the POST method.

1.1. Simple Subset of NETCONF Functionality

The framework and meta-model used for a RESTful API does not need to mirror those used by the NETCONF protocol. It just 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.

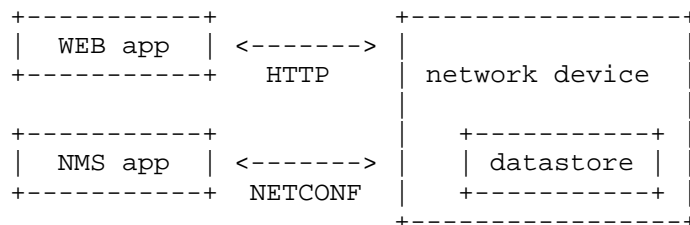
A simplified transaction model is needed that allows basic CRUD operations on a hierarchy of conceptual resources. This represents a limited subset of the transaction capabilities of the NETCONF protocol.

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

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

The RESTful API is not intended to replace NETCONF, but rather provide an additional simplified interface that follows RESTful principles and is compatible with a resource-oriented device abstraction. It is expected that applications that need the full feature set of NETCONF such as notifications will continue to use NETCONF.

The following figure shows the system components:



1.2. Data Model Driven API

YANG-API combines the simplicity of a RESTful API over HTTP with the predictability and automation potential of a schema-driven API.

A RESTful client using YANG-API will not use any data modelling language to define the application-specific content of the API. The client would discover each new child resource as it traverses the URIs return as Location IDs to discover the server capabilities.

This approach has 3 significant weaknesses wrt/ control of complex networking devices:

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

- o 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.

YANG-API 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. Note that the YANG modules and predictable URIs are optional to use by the client. They can be completely ignored without any loss of protocol functionality.

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.

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]:

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

- o startup configuration datastore
- o state data
- o user

1.3.2. HTTP

The following terms are defined in [RFC2616]:

- o entity tag
- o fragment
- o header line
- o 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/vnd.yang.api+xml" or "application/vnd.yang.api+json".
- o data resource: a resource with the media type "application/vnd.yang.data+xml" or "application/vnd.yang.data+json".
- o datastore resource: a resource with the media type "application/vnd.yang.datastore+xml" or "application/vnd.yang.datastore+json".
- o edit operation: a YANG-API operation on a data resource using the POST, PUT, PATCH, or DELETE method.
- o operation: the conceptual YANG-API operation for a message, derived from the method, request URI, headers, and message body.
- o operation resource: a resource with the media type "vnd.yang.operation+xml" or "vnd.yang.operation+json".
- o optional key: a key leaf for a YANG list data node, which MAY be omitted by the client when an instance of the list is created.
- o query parameter: a parameter (and its value if any), encoded within the query portion of the request URI.
- o resource: a conceptual object representing a manageable component within a device.
- o retrieval request: an operation using the GET or HEAD methods.
- o target resource: the resource that is associated with a particular message, identified by the "path" component of the request URI.

1.4. Overview

This document defines the YANG-API protocol, a RESTful API for accessing conceptual datastores containing data defined with YANG language. YANG-API provides an application framework and meta-model, using HTTP operations.

The YANG-API 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 and top-level data node resources available on the server.

1.4.1. Resource URI Map

The URI hierarchy for the YANG-API resources consists of an entry point and up to 4 top-level resources and/or fields. Refer to Section 5 for details on each URI.

```
/yang-api
  /datastore
    /<top-level-data-nodes> (config=true or false)
  /modules
    /module
  /operations
    /<custom protocol operations>
  /version
```

1.4.2. YANG-API Message Examples

The examples within this document use the non-normative example YANG module defined in Section 13.

This section shows some typical YANG-API message exchanges.

1.4.2.1. Retrieve the Top-level API Resource

By default, when a resource is retrieved, all of its fields are returned, but none (if any) of the nested resources are returned. Also, the default encoding is JSON. Data resources are encoded according to the encoding rules in [I-D.lhotka-netmod-json].

The client starts by retrieving the top-level API resource, using the entry point URI `"/yang-api"`.

```
GET /yang-api HTTP/1.1
Host: example.com
```

The server might respond as follows. The "module" lines below are split for display purposes only:

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

```
{
  "yang-api": {
    "modules": {
      "module": [
        "urn:ietf:params:xml:ns:yang:ietf-yang-api
          ?module=ietf-yang-api&revision=2012-05-27",
        "example.com?module=example-jukebox
          &revision=2012-05-30"
      ]
    },
    "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 /yang-api HTTP/1.1
Host: example.com
Accept: application/vnd.yang.api+xml
```

An alternate approach is provided using the "format" query parameter, as in this example request:

```
GET /yang-api?format=xml HTTP/1.1
Host: example.com
```

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/vnd.yang.api+xml
```

```
<yang-api>
  <modules> <!-- wrapped for display only -->
    <module>urn:ietf:params:xml:ns:yang:ietf-yang-api
      ?module=ietf-yang-api
      &revision=2012-05-27</module>
    <module>example.com?module=example-jukebox
      &revision=2012-05-30</module>
  </modules>
  <version>1.0</version>
</yang-api>
```

Refer to Section 3.3 for details on the GET operation.

1.4.2.2. Create New Data Resources

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

```
POST /yang-api/datastore/jukebox HTTP/1.1
Host: example.com
```

If the resource is created, the server might respond:

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

To create a new "artist" resource within the "jukebox" resource, the client might send the following request, Note that the arbitrary integer "index" is not provided, since it is an optional key:

```
POST /yang-api/datastore/jukebox/artist HTTP/1.1
Host: example.com
Content-Type: application/vnd.yang.data+json

{
  "artist" : {
    "name" : "The Foo Fighters"
  }
}
```

If the resource is created, the server might respond:

```
HTTP/1.1 201 Created
Date: Mon, 23 Apr 2012 17:02:00 GMT
Server: example-server
Location: http://example.com/yang-api/datastore/jukebox/artist/1
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,

```
POST /yang-api/datastore/jukebox/artist/1/album HTTP/1.1
Host: example.com
Content-Type: application/vnd.yang.data+json
```

```
{
  "album" : {
    "name" : "Wasting Light",
    "genre" : "example-jukebox:Alternative",
    "year" : 2012
  }
}
```

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/yang-api/datastore/
  jukebox/artist/1/album/Wasting%20Light
Last-Modified: Mon, 23 Apr 2012 17:03:00 GMT
ETag: b8389233a4c
```

Refer to Section 3.4 for details on the POST operation.

1.4.2.3. Replace an Existing Data Resource

Note: replacing a resource is a fairly drastic operation. The PATCH operation is often more appropriate.

The album sub-resource is re-added here for example purposes only. To replace the "artist" resource contents, the client might send:

```
PUT /yang-api/datastore/jukebox/artist/1 HTTP/1.1
Host: example.com
If-Match: b3830f23a4c
Content-Type: application/vnd.yang.data+json

{
  "artist" : {
    "name" : "Foo Fighters",
    "album" : {
      "name" : "Wasting Light",
      "genre" : "example-jukebox:Alternative",
      "year" : 2012
    }
  }
}
```

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
```

Refer to Section 3.5 for details on the PUT operation.

1.4.2.4. Patch an Existing Data Resource

To replace just the "year" field in the "album" resource, the client might send:

```
PATCH /yang-api/datastore/jukebox/artist/1/album/
      Wasting%20Light/year HTTP/1.1
Host: example.com
If-Match: b8389233a4c
Content-Type: application/vnd.yang.data+json

{ "year" : 2011 }
```

If the resource 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
```

Refer to Section 3.6 for details on the PATCH operation.

1.4.2.5. Delete an Existing Data Resource

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

```
DELETE /yang-api/datastore/jukebox/artist/1/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
```

Refer to Section 3.7 for details on the DELETE operation.

1.4.2.6. Invoke a Data Model Specific Operation

To invoke a data-model specific operation via an operation resource, the POST operation is used. A client might send a "backup-datastore" request as follows:

```
POST /yang-api/operations/backup-datastore HTTP/1.1  
Host: example.com
```

The server might respond:

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

Refer to Section 3.9 for details on using the POST operation with operation resources.

2. Framework

The YANG-API protocol defines a framework that can be used to implement a common API for configuration management. This section describes the components of the YANG-API framework.

2.1. Message Model

The YANG-API protocol uses HTTP entities for messages. A single HTTP message corresponds to a single protocol operation. A message can perform a single task on a single resource, such as retrieving a resource or editing a resource. It cannot be used to combine multiple tasks. The client cannot provide multiple (possibly unrelated) edit operations within a single request, like the NETCONF <edit-config> protocol operation.

2.2. Resource Model

The YANG-API 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 operations on that data. It can contain child nodes that are either "fields" or other resources. The child resource types and operations 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 fields and zero or more resources. A resource can be created and deleted independently of its parent resource, as long as the parent resource exist.

A field is a child node defined within a resource. A field can contain zero or more fields and zero or more resources. A field cannot be created and deleted independently of its parent resource.

All YANG-API resources and fields are defined in this document except datastore contents and protocol operations. These resource types are defined with YANG data definition statements and the "rpc" statement. A default mapping is defined to differentiate sub-resources from fields within data resources.

2.2.1. YANG-API Resource Types

The YANG-API protocol defines some application specific media types to identify each of the available resource types. The following table summarizes the purpose of each resource.

Resource	Media Type
API	application/vnd.yang.api
Datastore	application/vnd.yang.datastore
Data	application/vnd.yang.data
Operation	application/vnd.yang.operation

YANG-API Media Types

These resources are described in Section 5.

2.2.2. Resource Discovery

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

The YANG-API 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 operation to discover sub-resources within a particular resource.

Refer to Section 3.8.2 for more details on the "depth" parameter.

2.3. Datastore Model

A conceptual "unified datastore" is used to simplify resource management for the client. The YANG-API datastore is a combination of the running configuration and any non-configuration data supported by the device. By default only configuration data is returned by a GET operation on the datastore contents.

The underlying NETCONF datastores 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 YANG-API protocol. Transaction management and configuration persistence are handled by the server and not controlled by the client.

2.3.1. Content Model

The YANG-API protocol operates on a conceptual datastore defined with the YANG data modeling language. The server lists each YANG module it supports in the `"/yang-api/modules/module"` field in the top-level API resource type, using the YANG module capability URI format defined in RFC 6020.

The conceptual datastore contents and data-model-specific operations are identified by the set of YANG module capability URIs. All YANG-API content identified as either a data resource or an operation resource is defined with the YANG language.

The classification of data as configuration or non-configuration is derived from the YANG `"config"` statement. Data retrieval with the GET operation can be filtered in several ways, including the `"config"` parameter to retrieve configuration or non-configuration data.

The classification of data as a resource or field within a resource is derived from the rules specified in Section 5.3.2.

Data ordering behavior is derived from the YANG `"ordered-by"` statement. Editing mechanisms are provided to allow list or leaf-list resources to be inserted or moved in the same manner as NETCONF, and defined in YANG.

The server is not required to maintain system ordered data in any particular persistent order. The server **SHOULD** maintain the same data ordering for system ordered data until the next reboot or termination of the server.

2.3.2. Editing Model

The YANG-API datastore editing model is simple and direct, similar to the behavior of the `":writable-running"` capability in NETCONF.

Each YANG-API edit of a datastore resource is activated upon successful completion of the transaction. It is an implementation-specific matter how the server accomplishes a YANG-API edit request. For example, a server which only accepts edits through a candidate datastore may internally edit this datastore and perform the `"commit"` operation automatically.

Applications which need more control over the editing model might consider using NETCONF instead of YANG-API.

2.3.2.1. Edit Operation Discovery

Sometimes a server does not implement every operation for every resource. Sometimes data model requirements cause a node to implement a subset of the edit operations. For example, a server may not allow modification of a particular configuration data node after the parent resource has been created.

The OPTIONS operation can be used to identify which operations are supported by the server for a particular resource. For example, if the server will allow a data resource node to be created then the POST operation will be returned in the response.

2.3.2.2. Edit Collision Detection

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

- o 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.
- o 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.

Note that the server is only required to maintain these fields for a datastore resource, not for individual data resources.

Example:

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 replace a list entry with key value "11":

```
PATCH /yang-api/datastore/jukebox/artist/1/album/  
Wasting%20Light/year HTTP/1.1  
Host: example.com  
Accept: application/vnd.yang.data+json  
If-Unmodified-Since: Mon, 23 Apr 2012 17:01:00 GMT  
Content-Type: application/vnd.yang.data+json
```

```
{ "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/1.1 304 Not Modified
Date: Mon, 23 Apr 2012 19:01:00 GMT
Server: example-server
Last-Modified: Mon, 23 Apr 2012 17:45:00 GMT
ETag: b34aed893a4c
```

2.3.3. Locking Model

Datastore locking is not provided by YANG-API. An application that needs to make several changes to the running configuration datastore contents in sequence, without disturbance from other clients might consider using the NETCONF protocol instead of YANG-API.

2.3.4. Persistence Model

Each YANG-API 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.

Applications which need more control over the persistence model might consider using NETCONF instead of YANG-API.

2.3.5. Defaults Model

NETCONF has a rather complex defaults handling model for leafs. YANG-API attempts to avoid this complexity by restricting the operations that can be applied to a resource and fields within that resource.

The GET method returns only nodes that exist, which will be determined by the server. There is no mechanism for the client to ask the server for the default values that would be used for any nodes not present, but some default value is in use by the server. (There is no

retrieval mode like "with-defaults=report-all" in NETCONF.)

If a leaf definition has a default value, and the leaf has not been given a value yet, the server SHOULD NOT return any value for the leaf in the response for a GET operation.

Applications which need more control over the defaults model might consider using NETCONF instead of YANG-API.

2.4. Transaction Model

The YANG-API protocol does not provide a complex transaction model that allows for multiple protocol operations, or even operations on multiple resources in one protocol operation. A very simple "one operation one resource" per transaction model is used instead.

Applications which need more control over the transaction model might consider using NETCONF instead of YANG-API.

2.5. Extensibility Model

The YANG-API protocol is designed to be extensible for datastore content and data-model specific protocol operations. New protocol operations can be added without changing the entry point if they are optional and do not alter any existing operations.

Separate namespaces for each YANG module are used. Content encoded in XML will indicate the module using the "namespace" URI value in the YANG module. Content encoded in JSON will indicate the module using the module name specified in the YANG module. JSON encoding rules for module namespaces are specified in [I-D.lhotka-netmod-json].

2.6. Versioning Model

The version of a resource instance is identified with an entity tag, as defined by HTTP. The version identifiers in this section apply to the version of the schema definition of a resource. There are two types of schema versioning information used in the YANG-API protocol:

- o the YANG-API protocol version
- o data and operation resource definition versions

The protocol version is identified by the string used for the well-known URI entry point `/yang-api`. This would be changed (e.g., `/yang-api2`) if non-backward compatible changes are ever needed. Minor version changes that do not break backward-compatibility will not cause the entry point to change.

The API `"yang-api/version"` field can be used by the client to identify the exact version of the YANG-API protocol implemented by the server. This value will include the complete YANG-API protocol version. The `/yang-api` entry point will only change (e.g.,

"/yang-api2") if non-backward compatible changes are made to the protocol. The "/yang-api/version" field MUST be updated every time the protocol specification is republished.

The resource definition version for a data or operation resource is a date string, which is the revision date of the YANG module that defines the resource. The resource version for all other resource types is a numeric string, defined by the "/yang-api/version" field.

2.7. Retrieval Filtering Model

There are four types of filtering for retrieval of data resources in the YANG-API protocol.

- o conditional all-or-nothing: use some conditional test mechanism in the request headers and retrieve either a complete "200 OK" response if the condition is met, or a "304 Not Modified" Status-Line if the condition is not met.
- o data classification: request configuration or non-configuration data.
- o subset: request a subset of all possible instances of a list or leaf-list data resource.
- o filter: request a subset of all possible descendant nodes within the target resource. The "select" query parameter can be used for this purpose.

Refer to Section 5.3.4 for details on data retrieval filtering.

2.8. Access Control Model

The YANG-API protocol provides no granular access control for any content except for operation and data resources. The NETCONF Access Control Model (NACM) is defined in [RFC6536]. There is a specific mapping between YANG-API operations and NETCONF edit operations, defined in Table 1. The resource path also 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 YANG-API messages.

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

3. Operations

The YANG-API protocol uses HTTP methods to identify the CRUD operation requested for a particular resource or field within a resource. The following table shows how the YANG-API operations relate to NETCONF protocol operations:

YANG-API	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 Operations in YANG-API

The NETCONF "remove" operation attribute is not supported by the HTTP DELETE method. The resource must exist or the DELETE operation will fail.

This section defines the YANG-API protocol usage for each HTTP method.

3.1. OPTIONS

The OPTIONS method is sent by the client to discover which methods are supported by the server for a specific resource, or field within a resource. It is supported for all media types. Note that implementation of this operation is part of HTTP, and this section does not introduce any additional requirements.

The request MUST contain a request URI that contains at least the entry point component.

The server will return a "Status-Line" header containing "204 No Content". and include the "Allow" header in the response. This header will be filled in, based on the target resource media type. Other headers MAY also be included in the response.

Example 1:

A client might request the methods supported for a data resource called "library"

```
OPTIONS /yang-api/datastore/jukebox/library HTTP/1.1
Host: example.com
```

The server might respond (for a config=true list):

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:01:00 GMT
Server: example-server
Allow: OPTIONS,HEAD,GET,POST,PUT,PATCH,DELETE
```

Example 2:

A client might request the methods supported for a non-configuration leaf within a data resource:

```
OPTIONS /yang-api/datastore/jukebox/library/
        song-count HTTP/1.1
Host: example.com
```

The server might respond:

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:02:00 GMT
Server: example-server
Allow: OPTIONS,HEAD,GET
```

Example 3:

A client might request the methods supported for an operation resource called "play":

```
OPTIONS /yang-api/operations/play HTTP/1.1
Host: example.com
```

The server might respond:

```
HTTP/1.1 204 No Content
Date: Mon, 23 Apr 2012 17:02:00 GMT
Server: example-server
Allow: POST
```

3.2. HEAD

The HEAD operation is sent by the client to retrieve just the headers that would be returned for the comparable GET operation, without the response body. The HTTP HEAD method is used for this operation. 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 operation are supported by the HEAD operation. For example, the "select" query parameter can be used to specify a field within the target resource.

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.

Example:

The client might request the response headers for the default (JSON) representation of the "library" resource:

```
HEAD /yang-api/datastore/jukebox/library HTTP/1.1
Host: example.com
```

The server might respond:

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

3.3. GET

The GET operation is sent by the client to retrieve data and meta-data for a resource or field within a resource. The HTTP GET method is used for this operation. 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 following query parameters are supported by the GET operation:

Name	Section	Description
config	3.8.1	Request either configuration or non-configuration data
depth	3.8.2	Control the depth of a retrieval request
format	3.8.3	Request either JSON or XML content in the response


```
| select | 3.8.6 | Specify a field within the target resource |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

GET Query Parameters

The server **MUST NOT** return any data resources or fields within 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 /yang-api/datastore/jukebox/library/artist/
    1/album HTTP/1.1
Host: example.com
```

The server might respond:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 17:02:40 GMT
Server: example-server
Content-Type: application/vnd.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
  }
}
```

3.4. POST

The POST operation is sent by the client for various reasons. The HTTP POST method is used for this purpose. The request MUST contain a request URI that contains a target resource that identifies one of the following resource types:

Type	Description
Data	Create a configuration data resource
Operation	Invoke protocol operation
Transaction	Create a new transaction

Resource Types that Support POST

The following query parameters are supported by the POST operation:

Name	Section	Description
insert	3.8.4	Specify where to insert a resource
point	3.8.5	Specify the insert point for a resource

POST Query Parameters

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

If the user is not authorized to invoke the target (operation) resource, or 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.

3.5. PUT

The PUT operation is sent by the client to replace the target resource.

The HTTP PUT method is used for this purpose. The request MUST contain a request URI that contains a target resource that identifies the data resource to replace.

The following query parameters are supported by the PUT operation:

Name	Section	Description
insert	3.8.4	Specify where to move a resource
point	3.8.5	Specify the move point for a resource

PUT Query Parameters

If the PUT operation succeeds, a "200 OK" Status-Line is returned, and there is no response message body.

If the user is not authorized to 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.

3.6. PATCH

The PATCH operation uses the HTTP PATCH method defined in [RFC5789] to provide a "merge" editing mode for data resources. Instead of replacing all or part of the target resource, the supplied values are merged into the target resource.

If the PATCH operation succeeds, a "200 OK" Status-Line is returned, and there is no response message body.

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.

3.7. DELETE

The DELETE operation uses the HTTP DELETE method to delete the target resource.

If the DELETE operation succeeds, a "200 OK" 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.

3.8. Query Parameters

Each YANG-API operation allows zero or more query parameters to be present in the request URI. Refer to Section 3 for details on the query parameters used in the definition of each operation.

Query parameters can be given in any order. Each parameter can appear zero or one time. A default value may apply if the parameter is missing.

This section defines all the YANG-API query parameters.

3.8.1. "config" Parameter

The "config" parameter is used to specify whether configuration or non-configuration data is requested.

This parameter is only supported for the GET and HEAD methods. It is also only supported if the target resource is a data resource.

```
syntax: config= true | false
default: true
```

Example:

This example request by the client would retrieve only the non-configuration data nodes that exist within the second-level "library" resource.

```
GET /yang-api/datastore/jukebox/library?config=false HTTP/1.1
Host: example.com
Accept: application/vnd.yang.data+xml
```

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/vnd.yang.data+json
```

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

```
}
```

3.8.2. "depth" Parameter

The "depth" parameter is used to specify the number of nest levels returned in a response for a GET operation. A nest-level consists of the target resource and any child nodes which are optional data nodes (anyxml, leaf, or leaf-list). A non-presence container is transparent when determining the nest level. A child node (which is not a non-presence container) within a non-presence container is used to determine the nest-level.

The start level is determined by the target resource for the operation.

```
syntax: depth=<range: 1..max> | unbounded
default: 1
```

Example:

This example operation would retrieve 2 levels of configuration data nodes that exist within the top-level "jukebox" resource.

```
GET /yang-api/datastore/jukebox?depth=2 HTTP/1.1
Host: example.com
Accept: application/vnd.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/vnd.yang.data+json
```

```
{
  "jukebox" : {
    "library" : {
      "artist" : {
        "index" : 1,
        "name" : "Foo Fighters"
      }
    },
    "player" : {
      "gap" : 0.5
    }
  }
}
```

```
}
```

3.8.3. "format" Parameter

The "format" parameter is used to specify the format of any content returned in the response. Note that the "Accept" header MAY be used instead of this parameter to identify the format desired in the response. For example:

```
GET /yang-api/datastore/routing HTTP/1.1
Host: example.com
Accept: application/vnd.yang.data+xml
```

This example request would retrieve only the configuration data nodes that exist within the top-level "routing" resource, and retrieve them in XML encoding instead of JSON encoding.

The "format" parameter is only supported for the GET and HEAD methods. It is supported for all YANG-API media types.

```
syntax: format= xml | json
default: json
```

Example:

```
GET /yang-api/datastore/routing?format=xml HTTP/1.1
Host: example.com
```

This example URI would retrieve only the configuration data nodes that exist within the top-level "routing" resource, and retrieve them in XML encoding instead of JSON encoding.

3.8.4. "insert" Parameter

The "insert" parameter is used to specify how a resource should be inserted (or moved) within the user-ordered list or leaf-list data resource.

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, not the system.

If the values "before" or "after" are used, then a "point" parameter for the insertion parameter MUST also be present.

```
syntax: insert= first | last | before | after
default: last
```

Example:

Request from client:

```
POST /yang-api/datastore/jukebox/library/artist/1/album
     /Wasting%20Light/song?insert=first HTTP/1.1
Host: example.com
Content-Type: application/vnd.yang.data+json

{
  "song" : {
    "name" : "Bridge Burning",
    "location" : "/media/bridge_burning.mp3",
    "format" : "MP3",
    "length" : 286
  }
}
```

Response from server: 201 status

```
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/yang-api/datastore/jukebox
         /library/artist/1/album?Wasting%20Light/song/1
ETag: eeeada438af
```

3.8.5. "point" Parameter

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. It is ignored unless the "insert" query parameter is also present, and has the value "before" or "after".

This parameter contains the instance identifier of the resource, or field within a resource, to be used as the insertion point for a POST or PUT operation. It is encoded according to the rules defined in Section 5.3.1. There is no default for this parameter.

syntax: point= <instance-identifier of insertion point node>

Example:

In this example, the client is moving an existing "song" resource within an "album" resource after another song. The request URI is split for display purposes only.

Request from client:

```
PUT /yang-api/datastore/jukebox/library/artist/1/album/  
Wasting%20Light/song/2?insert=after  
&point=/yang-api/datastore/jukebox/library/artist/1/  
album/Wasting%20Light/song/4 HTTP/1.1  
Host: example.com
```

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
```

3.8.6. "select" Parameter

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 a relative path expression, using the target resource as the context node.

It is supported for all resource types except operation resources. The contents are encoded according to the "api-select" rule defined in Section 5.3.1. This parameter is only allowed for GET and HEAD operations.

[FIXME: the syntax of the select string is still TBD; XPath, schema-identifier, regular expressions, something else]

Refer to Section 1.4.2 for example request messages using the "select" parameter.

3.9. Protocol Operations

The YANG-API also allows data-model specific protocol operations to be invoked using the POST method. The media type "vnd.yang.operation+xml" or "vnd.yang.operation+json" MUST be used in the "Content-Type" field in the message header.

Data model specific operations are supported. The syntax and semantics of these operations exactly correspond to the YANG "rpc" statement definition for the operation.

Any input for a protocol operation is encoded in an element called "input", which corresponds to the <input> element in a NETCONF message. The child nodes of the "input" element are encoded

according to the data definition statements in the input section of the "rpc" statement.

Any output for a protocol operation is encoded in an element called "output", which corresponds to the <rpc-reply> element in a NETCONF message. The child nodes of the "output" element are encoded according to the data definition statements in the output section of the "rpc" statement.

4. Messages

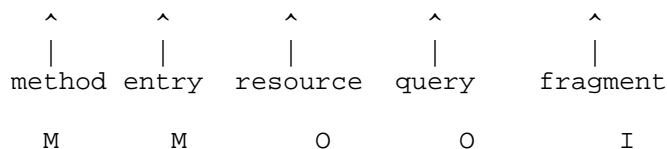
This section describes the messages that are used in the YANG-API protocol.

4.1. Request URI Structure

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

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

`<OP> /yang-api/<path>?<query>#<fragment>`



M=mandatory, O=optional, I=ignored

`<text>` replaced by client with real values

- o method: the HTTP method identifying the YANG-API operation requested by the client, to act upon the target resource specified in the request URI. YANG-API operation details are described in Section 3.
- o entry: the well-known YANG-API entry point ("/yang-api").
- 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 "vnd.yang.api".
- o query: the set of parameters associated with the YANG-API 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.
- o fragment: This field is not used by the YANG-API protocol.

The client SHOULD NOT assume the final structure of a URI path for a

resource. Instead, existing resources can be discovered with the GET operation. 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.

4.2. Message Headers

There are several HTTP header lines utilized in YANG-API 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 3 for examples on how particular headers are used.

There are some request headers that are used within YANG-API, usually applied to data resources. The following tables summarize the headers most relevant in YANG-API 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-Range	Only retrieve range if resource unchanged
If-Unmodified-Since	Only perform the action if un-modified since time
Range	Specify a range of data resource entries

YANG-API Request Headers

The following tables summarize the headers most relevant in YANG-API message responses:

Name	Description
Allow	Valid actions when 405 error returned
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

YANG-API Response Headers

4.3. Message Encoding

YANG-API messages are encoded in HTTP according to RFC 2616. The "utf-8" character set is used for all messages. YANG-API 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. XML attributes are not used and will be ignored if present in an XML-encoded message.

JSON encoding rules are defined in [I-D.lhotka-netmod-json]. Special encoding rules are needed to handle multiple module namespaces and provide consistent data type processing.

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

Response output content encoding format is identified with the Accept header, the "format" query parameter, or if neither is specified, the request input encoding format is used. If there was no request input, then the default output encoding is JSON. File extensions encoded in the request are not used to identify format encoding.

4.4. 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.

4.5. Message Caching

Since the datastore contents change at unpredictable times, responses from a YANG-API 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 headers such as "If-None-Match" or "If-Modified-Since" 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 operation 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.

5. Resources

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

5.1. API Resource (/yang-api)

The API resource contains the state and access points for the YANG-API features.

It is the top-level resource and has the media type "application/vnd.yang.api+xml" or "application/vnd.yang.api+json". It is accessible through the well-known URI "/yang-api".

This resource has the following fields:

Field Name	Description
datastore	Link to "datastore" resource
modules	YANG module capability URIs
operations	Data-model specific operations

YANG-API Resource Fields

5.1.1. /yang-api/datastore

This mandatory resource represents the running configuration datastore and any non-configuration data available. It may be retrieved and edited directly. It cannot be created or deleted by the client. This resource type is defined in Section 5.2.

5.1.2. /yang-api/modules

This mandatory field contains the identifiers for the YANG data model modules supported by the server. There MUST be exactly one instance of this field.

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

5.1.3. /yang-api/operations

This optional field provides access to the data-model specific protocol operations supported by the server. The server MAY omit

this field if no data-model specific operations are advertised.

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

5.1.3.1. /yang-api/modules/module

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

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" field MUST NOT be used instead.

The contents of this field are encoded with the "uri" derived type from the "ietf-iana-types" modules in [RFC6021].

There are additional encoding requirements for this field. The URI MUST follow the YANG module capability URI formatting defined in section 5.6.4 of [RFC6020].

5.1.3.2. Retrieval Example

In this example the client is retrieving the modules field from the server in the default JSON format:

```
GET /yang-api?select=modules HTTP/1.1
Host: example.com
Accept: application/vnd.yang.api+json
```

The server might respond as follows. Note that the content below is split across multiple lines for display purposes only:

```
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/vnd.yang.api+json
```

```
{
  "yang-api": {
    "modules": {
      "module": [
        "example.com?module=foo&revision=2012-01-02",
        "example.com?module=bar&revision=2011-10-10",
        "example.com?module=itf&revision=2011-10-10
          &feature=restore"
      ]
    }
  }
}
```

5.1.4. /yang-api/version

This mandatory field identifies the specific version of the YANG-API protocol implemented by the server.

The same server-wide response MUST be returned each time this field is retrieved. It is assigned by the server when the server is started. The server MUST return the value "1.0" for this version of the YANG-API protocol.

This field is encoded with the rules for an "enumeration" data type, using the following leaf definition:

```
leaf version {
  config false;
  type enum {
    enum "1.0" {
      description
        "Version 1.0 of the YANG-API protocol.";
    }
  }
}
```


5.2. Datastore Resource

A datastore resource represents the conceptual root of a tree of data resources.

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.

The server **SHOULD** 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 **SHOULD** be changed to a new previously unused value if changes to any configuration data resources within the datastore are made.

A datastore resource can be retrieved with the GET operation, to retrieve either configuration data resources or non-configuration data resources within the datastore. The "config" query parameter is used to choose between them. Refer to Section 3.8.1 for more details.

The depth of the subtrees returned in retrieval operations can be controlled with the "depth" query parameter. The number of nest levels, starting at the target resource, can be specified, or an unlimited number can be returned. Refer to Section 3.8.2 for more details.

A datastore resource cannot be written directly with any edit operation. Only the configuration data resources within the datastore resource can be edited.

5.3. 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.

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. The resource entity tag **SHOULD** be changed to a new previously unused value if changes to the resource or any configuration field within the resource is altered.

A data resource can be retrieved with the GET operation, to retrieve either configuration data resources or non-configuration data resources within the target resource. The "config" query parameter is used to choose between them. Refer to Section 3.8.1 for more details.

The depth of the subtrees returned in retrieval operations can be controlled with the "depth" query parameter. The number of nest levels, starting at the target resource, can be specified, or an unlimited number can be returned. Refer to Section 3.8.2 for more details.

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 3 for more details on edit operations.

5.3.1.1. Encoding YANG Instance Identifiers in the Request URI

In YANG, data nodes are named with an absolute XPath expression, from the document root to the target resource. In YANG-API, URL friendly path expressions are used instead.

The YANG "instance-identifier" (i-i) data type is represented in YANG-API 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

YANG-API instance-identifier Type Conversion

The "path" component of the request URI contains the absolute path expression that identifies the target resource. The "select" query parameter is used to optionally identify the requested data nodes within the target resource to be retrieved in a GET operation.

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 YANG-API 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 5.3.1.1. 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 the "select" is present, it is encoded, starting with a child node of the target resource, according to the "api-select" rule defined in Section 5.3.1.1.

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 operation to create a new data resource for a list node does not allow the 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 5.3.1.1, 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 leaves.
- 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].

Notifications are not supported by YANG-API because they are not supported by HTTP. YANG notification statements are ignored by a YANG-API server.

Examples:

```
/yang-api/datastore/jukebox/library/artist/17&select=name
/yang-api/datastore/newlist/17&select=nextlist/22/44/myleaf
/yang-api/datastore/somelist/fred%20and%20wilma
/yang-api/datastore/somelist/fred%20and%20wilma/address
```

5.3.1.1. ABNF For Data Resource Identifiers

The following ABNF syntax is used to construct YANG-API path identifiers:

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

[FIXME: the syntax for the select string is still TBD]
api-select = 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>
```

5.3.2. Identifying YANG-defined Data Resources

The data resources used in YANG-API are defined with YANG data definition statements.

Not every data node defined in a YANG module should be treated as a resource. The YANG-API needs to know which YANG data nodes are resources, and which are fields within a resource.

For data resources, YANG-API uses a simple algorithm for defining resource boundaries, within the conceptual sub-trees described by YANG data definition statements.

All top-level data nodes are considered to be resources. For nodes within a top-level resource:

- o a presence container starts a new resource
- o a list starts a new resource
- o an optional terminal node (anyxml, leaf, or leaf-list) starts a new resource
- o a data node of type "anyxml" cannot have any sub-resources

A non-configuration data node cannot be a separate resource from its parent. Only top-level data nodes are considered to be resources (which only support retrieval methods).

5.3.3. Identifying Optional Keys

It is sometimes useful to have the server assign the key(s) for a new resource. The "Location" header will indicate the key value(s) that the server selected, so the client does not need to provide all the key leaf values.

It is useful to identify in the YANG data model module which key leafs are optional to provide, and which are not. The YANG extension statement "optional-key" is provided to indicate that the leaf definition represents an optional key.

The client MAY provide a value for a key leaf in a POST operation. Refer to Section 8 for details on the "optional-key" extension. Refer to Section 13 for usage examples of this YANG extension statement.

5.3.4. Data Resource Retrieval

There are four types of filtering for retrieval of data resources. This section defines each mode.

5.3.4.1. Conditional Retrieval

The HTTP headers (such as "If-Modified-Since" and "If-Match") can be used in for a request message for a GET operation to check a condition within the server state, such as the last time the datastore resource was modified, or the resource entity tag of the target resource.

If the condition is met according to the header definition, a "200 OK" Status-Line and the data requested is returned in the response

message. If the condition is not met, a "304 Not Modified" Status-Line is returned in response message instead.

5.3.4.2. Data Classification Retrieval

The "config" query parameter can be used with the GET operation to specify whether configuration or non-configuration data is requested. Refer to Section 3.8.1 for more details on the "config" query parameter.

5.3.4.3. Subset Retrieval

The "Range" header is used to request a specific subset of the instances of a list or leaf-list data resource that are returned by the server for a retrieval operation. Normally, if the target resource in a request message does not specify an instance, then all instances are returned.

The YANG-API protocol uses the token "entries" instead of "bytes" as the range units.

The entries are numbered starting from "0". A list or leaf-list can change order between requests so the client needs to be aware of the data model semantics, and whether the list contents are stable enough to use the subset retrieval mechanism.

If the requested range cannot be returned because the range specification includes index values for entries that do not exist, then an error occurs, and the server MUST return a "416 Requested range not satisfiable" Status-Line.

If the range request can be satisfied, then a "200 OK" Status-Line is returned, and the response MUST include a "Content-Range" header indicating which entries are returned. The response message body contains the data for the requested range of entries.

Example:

In this example, the client is requesting 5 "artist" resource entries, starting with the 10th entry:

Request from client:

```
GET /yang-api/datastore/jukebox/library/artist HTTP/1.1
Host: example.com
Accept: application/vnd.yang.data+json
Range: entries 10-14
```

Response from server:

```
HTTP/1.1 200 OK
Date: Mon, 23 Apr 2012 13:01:20 GMT
Cache-Control: no-cache
Pragma: no-cache
Content-Type: application/vnd.yang.data+json
Content-Range: entries 10-14
Server: example-server
Last-Modified: Mon, 23 Apr 2012 02:12:20 GMT
ETag: abcada438af
```

```
{
  "artist" : {
    // content removed for brevity
  }
}
```

5.3.4.4. Filtered Retrieval

The "select" query parameter is used to specify a filter that should be applied to the target resource to request a subset of all possible descendant nodes within the target resource.

The format of the "select" parameter string is defined in Section 3.8.6. The set of nodes selected by the filter expression is applied to each context node identified by the target resource.

5.4. 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 /yang-api/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.

5.4.1. 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 /yang-api/datastore/operations/reboot HTTP/1.1
Host: example.com
Content-Type: application/vnd.yang.data+json
```

```
{
  "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
```

5.4.2. 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 {
  input {
    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 /yang-api/datastore/operations/get-reboot-info HTTP/1.1
Host: example.com
```

The server might respond:

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

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

5.4.3. Identifying YANG-defined Operation Resources

The operation resources used in YANG-API are defined with YANG "rpc" statements. All "rpc" statements within a YANG module that are supported by the server are available as operation resources.

6. Error Reporting

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

The following table summarizes the return status codes used specifically by YANG-API 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
413 Request Entity Too Large	too-big error
414 Request-URI Too Large	too-big error
415 Unsupported Media Type	non YANG-API media type
416 Requested range not satisfiable	If-Range error
500 Internal Server Error	operation-failed
501 Not Implemented	unknown-operation
503 Service Unavailable	Recoverable server error

HTTP Status Codes used in YANG-API

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 following YANG data definition statement:

```
container errors {
  config false;

  list error {
    reference "RFC 6241, Section 4.3";
    leaf error-type {
      mandatory true;
      type enumeration {
        enum transport;
        enum rpc;
        enum protocol;
        enum application;
      }
    }
    leaf error-tag {
      mandatory true;
      type string;
    }
    leaf error-app-tag {
      type string;
    }
    leaf error-path {
      type string; // YANG-API encoded instance-identifier
    }
    leaf error-message {
      type string;
    }
    container error-info {
      // anyxml content here
    }
  }
}
```

Example:

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

```
POST /yang-api/operations/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/vnd.yang.api+json

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

7. RelaxNG Grammar

TBD

8. YANG-API module

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

```
<CODE BEGINS> file "ietf-yang-api@2012-11-30.yang"
```

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

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

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

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

  description
    "This module contains a collection of YANG language extensions
    to describe REST API Resources using YANG data definition
    statements.

    Copyright (c) 2012 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-api-01.txt

  // RFC Ed.: update the date below with the date of RFC publication
  // and remove this note.
  revision 2012-11-30 {
```



```
    description
      "Initial revision.";
    reference
      "RFC XXXX: YANG-API Protocol.";
  }

  /*
   * Extensions
   */

  extension optional-key {
    description
      "This extension is used to allow the client to create
       a new instance of a resource without providing a
       value for the key leaf containing this statement.
       This extension is ignored for NETCONF, and only
       applies to YANG-API resources and fields.
       This extension is ignored unless it appears
       directly within a 'leaf' data definition statement.";
  }
}

<CODE ENDS>
```

9. IANA Considerations

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-api
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-api
namespace: urn:ietf:params:xml:ns:yang:ietf-yang-api
prefix: api
reference: RFC XXXX

10. Security Considerations

TBD

11. Change Log

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

11.1. 00-01

- o expanded introduction
- o removed transactions
- o removed capabilities
- o simplified editing model
- o removed global protocol operations from ietf-yang-api.yang
- o changed RPC operation terminology to protocol operation
- o updated JSON draft reference
- o updated open issues section
- o updated IANA section

12. Open Issues

- o Which WG should do this work? NETCONF? NETMOD? It is not clear since YANG-API builds on concepts and standards from documents owned by both working groups.
- o Resource creation order and other dependencies between resources are not well identified in YANG. YANG has leafrefs and instance-identifiers, which can be used to identify some order dependencies. Are any new mechanisms needed in YANG-API needed to identify resource creation order and other dependency requirements?
- o There is no "message-id" field in a YANG-API 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?
- o Should sessions be used or not? Should "reusable sessions" be used? Better for auditing? How does locking of the /yang-api/datastore resource work for multiple edits if a session is 1 operation? When does the server release the lock and decide it has been abandoned or client was disconnected?
- o What syntax should be used for the "select" query parameter?
- o Should the "/yang-api/modules" field within the API resource be a separate resource, with its own timestamp? Currently the API timestamp is coupled to any changes to the list of loaded modules. Should the API resource be static and cacheable?
- o What to do about no REMOVE operation, just DELETE? The effect is local to the request; in a NETCONF edit-config it is worse, since the netconf request might create/delete/modify many nodes
- o Should every YANG data node be a data resource and every YANG RPC statement an operation resource? Is a YANG extension needed to allow data modeler control of resource boundaries?
- o Encoding of leafrefs? Is there some additional meta-data needed? Do leafref nodes need to be identified in responses (RFC 5988) or is the YANG module definition sufficient to provide this meta-data?
- o What should the default algorithm be for defining data resources? Should the default for an augment from another namespace be to start a new resource? Top-level data node defaults as a resource OK?

- o Is the token "entries" legal in the YANG-API usage of Range? What units should be used? "bytes" is the only token defined by HTTP.
- o Are all header lines used by YANG-API 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?
- o Should the <errors> element returned in error responses be a separate media type?
- o How should additional datastores be supported, which may be added to the NETCONF/NETMOD framework in the future?

13. Example YANG Module

```
module example-jukebox {

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

    import ietf-yang-api { prefix api; }

    organization "Example, Inc.";
    description "Example Jukebox Data Model Module";
    revision "2012-05-30";

    identity genre {
        description "Base for all genre types";
    }

    // abbreviated list of genre classifications
    identity Alternative {
        base genre;
    }
    identity Blues {
        base genre;
    }
    identity Country {
        base genre;
    }
    identity Jazz {
        base genre;
    }
    identity Pop {
        base genre;
    }
    identity Rock {
        base genre;
    }

    container jukebox {
        presence
            "An empty container indicates that the jukebox
            service is available";

        container library {
            list artist {
                key index;
                unique name;
            }
        }
    }
}
```

```
leaf index {
  api:optional-key;
  type uint32;
  description
    "Optional key used instead of natural key for
     example. Also rare but possible artists with
     the same name are really different entities.";
}
leaf name {
  type string;
}

list album {
  key name;
  leaf name {
    type string {
      length "1 .. max";
    }
  }
  leaf genre {
    type identityref { base genre; }
  }
  leaf year {
    type uint16 {
      range "1900 .. max";
    }
  }
}
list song {
  api:optional-key;
  key index;
  ordered-by user;
  leaf index {
    type uint32;
  }
  leaf name {
    mandatory true;
    type string;
  }
  leaf location {
    mandatory true;
    type string;
  }
  leaf format {
    type string;
  }
  leaf length {
    units "seconds";
    type uint32;
  }
}
```



```
    }  
  }  
}  
leaf artist-count {  
  config false;  
  type uint32;  
  units "songs";  
  description "Number of artists in the library";  
}  
leaf album-count {  
  config false;  
  type uint32;  
  units "albums";  
  description "Number of albums in the library";  
}  
leaf song-count {  
  type uint32;  
  units "songs";  
  description "Number of songs in the library";  
}  
}  
  
list playlist {  
  description  
    "Example configuration data resource";  
  key name;  
  leaf name {  
    type string;  
  }  
  leaf description {  
    type string;  
  }  
  list song {  
    description  
      "Example nested configuration data resource";  
    ordered-by user;  
    key index;  
    leaf index {  
      api:optional-key;  
      type uint32;  
    }  
    leaf id {  
      mandatory true;  
      type instance-identifier;  
      description  
        "Song identifier. Must identify an instance of  
        /jukebox/library/artist/album/song."  
    }  
  }  
}
```

```
        The id is not the key to allow duplicates
        in a playlist";
    }
}

container player {
  leaf gap {
    description "Time gap between each song";
    units "tenths of seconds";
    type decimal64 {
      fraction-digits 1;
      range "0.0 .. 2.0";
    }
  }
}

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";
    }
  }
}
```

14. Normative References

- [I-D.lhotka-netmod-json]
Lhotka, L., "Modeling JSON Text with YANG",
draft-lhotka-netmod-yang-json-00 (work in progress),
October 2012.
- [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.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
Resource Identifier (URI): Generic Syntax", STD 66,
RFC 3986, January 2005.
- [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.
- [RFC6021] Schoenwaelder, J., "Common YANG Data Types", RFC 6021,
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.

Authors' Addresses

Andy Bierman
YumaWorks

Email: andy@yumaworks.com

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

NETCONF Working Group
Internet-Draft
Updates: 4253 (if approved)
Intended status: Standards Track
Expires: January 02, 2015

K. Watsen
Juniper Networks
July 2014

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

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

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

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

This Internet-Draft will expire on January 02, 2015.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Requirements Terminology	2
2. Introduction	2
2.1. Applicability Statement	3
2.2. Update to RFC 4253	3
2.3. Draft Naming	3
3. Benefits to Device Management	3
4. Protocol	5
5. SSH Server Identification and Verification	5
6. Device Configuration	6
7. Security Considerations	7
8. IANA Considerations	8
9. Acknowledgements	8
10. References	8
10.1. Normative References	8
10.2. Informative References	9
Appendix A. Change Log	9
A.1. 05 to 06	9
A.2. 04 to 05	10
A.3. 03 to 04	10
A.4. 02 to 03	11
A.5. 01 to 02	11
A.6. 00 to 01	11

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 apparent omission 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:

- o The network element may proactively call home after being powered on for the first time to register itself with its management system.
- o 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 address to a mapping service.
- o The network element may be configured in "stealth mode" and thus doesn't have any open ports for the management system to connect to.
- o The network element may be deployed behind a firewall that doesn't allow SSH access to the internal network.
- o 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.
- o 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)

- o The NETCONF server initiates a TCP connection to the NETCONF client on the IANA-assigned SSH for NETCONF Call Home port YYYY.
- o The TCP connection is accepted and a TCP session is established.
- o 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]).
- o The SSH connection is established.

The NETCONF client's perspective (e.g., the management system)

- o The NETCONF client listens for TCP connections on the IANA-assigned NETCONF over SSH Call Home port YYYY.
- o The NETCONF client accepts an incoming TCP connection and a TCP session is established.
- o 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]).
- o 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 possess 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 grammar, 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

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

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

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

This Internet-Draft will expire on October 12, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Connection Initiation	3
3. Message Framing	3
4. Connection Closure	3
5. Certificate Validation	3
6. Server Identity	4
7. Client Identity	4
8. Cipher Suites	6
9. Security Considerations	6
10. IANA Considerations	7
11. Acknowledgements	7
12. References	8
12.1. Normative References	8
12.2. Informative References	8
Appendix A. Changes from RFC 5539	9
Authors' Addresses	9

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 explicitly 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. Similarly, 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/>

Network Working Group
Internet Draft

Intended status: Experimental
Expires: April 2016

T. Mizrahi
Y. Moses
Technion, Israel Institute of Technology
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

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

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

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

The list of current Internet-Drafts can be accessed at
<http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at
<http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on April 15, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction.....	3
2. Conventions used in this document.....	3
2.1. Key words.....	3
2.2. Abbreviations.....	4
2.3. Terminology.....	4
3. Using Time in NETCONF.....	4
3.1. The Time Capability in a Nutshell.....	4
3.2. Notifications and Cancellation Messages.....	6
3.3. Synchronization Aspects.....	8
3.4. Scheduled Time Format.....	9
3.5. Scheduling Tolerance.....	9
3.6. Near Future Scheduling vs. Far Future Scheduling.....	10
3.7. Time Interval Format.....	12
4. Time Capability.....	13
4.1. Overview.....	13
4.2. Dependencies.....	13
4.3. Capability Identifier.....	13
4.4. New Operations.....	13
4.5. Modifications to Existing Operations.....	14
4.5.1. Affected Operations.....	14
4.5.2. Processing Scheduled Operations.....	15
4.6. Interactions with Other Capabilities.....	15
5. sched-max-futures.....	16
5.1. <scheduled-time> Example.....	16
5.2. <get-time> Example.....	17
5.3. Error Example.....	17
6. Security Considerations.....	18
6.1. General Security Considerations.....	18
6.2. YANG Module Security Considerations.....	19
7. IANA Considerations.....	20
8. Acknowledgments.....	20
9. References.....	21
9.1. Normative References.....	21
9.2. Informative References.....	21
Appendix A. YANG Module for the Time Capability.....	22

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, T_1 , T_2 , ..., T_n , 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 T_s in Figure 1. The server then executes the RPC at the scheduled time T_s , 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, T_e .

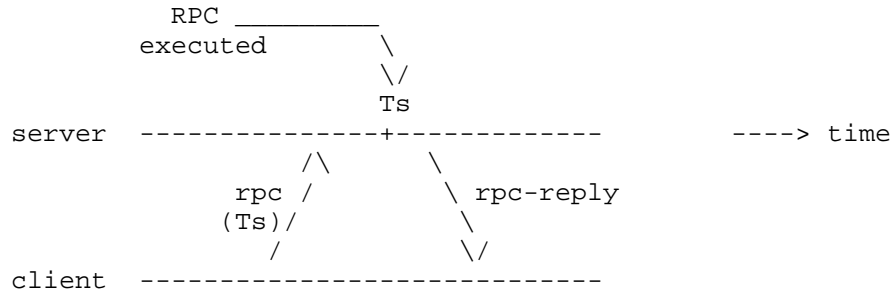


Figure 1 Scheduled RPC

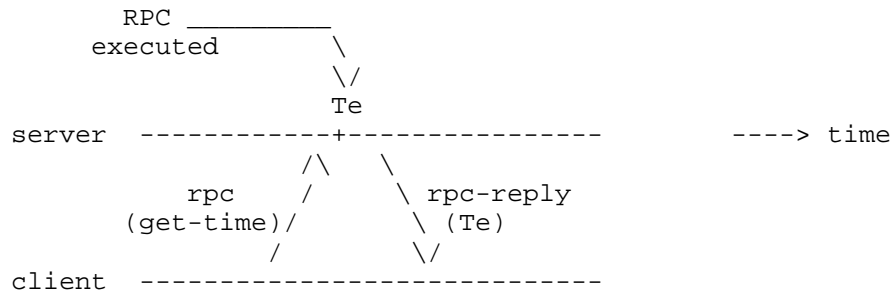


Figure 2 Reporting the Execution Time of an RPC

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 T_s , for example 5 seconds in the future, causing the servers to invoke the commit as close as possible to time T_s .

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, T_s , as well as the get-time element. This allows the client to receive feedback about the actual execution time, T_e . Ideally, $T_s = T_e$. However, the server may execute the RPC at a slightly different time than T_s , for example if the server is tied up with other tasks at T_s .

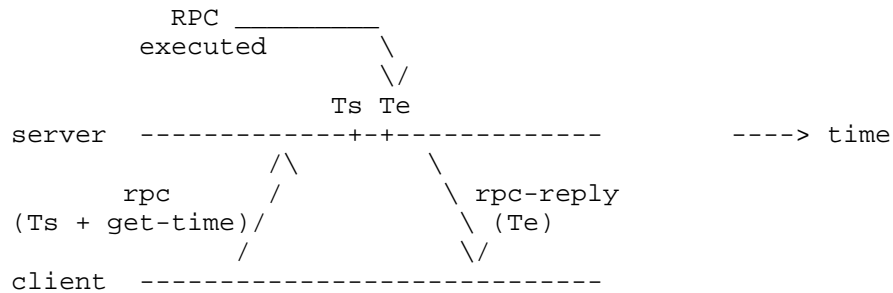


Figure 3 Scheduling and Reporting

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}.

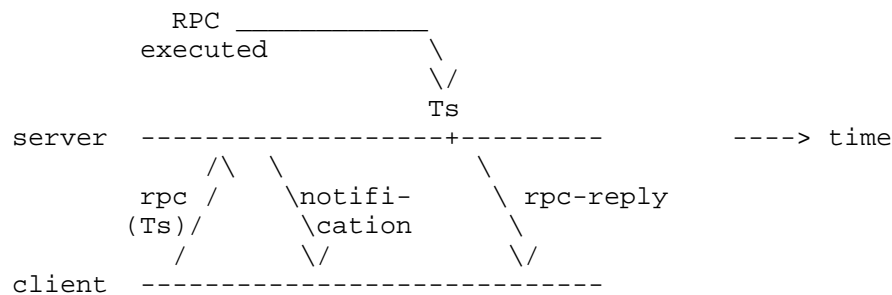


Figure 4 Scheduled RPC with Notification

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 T_s . 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.

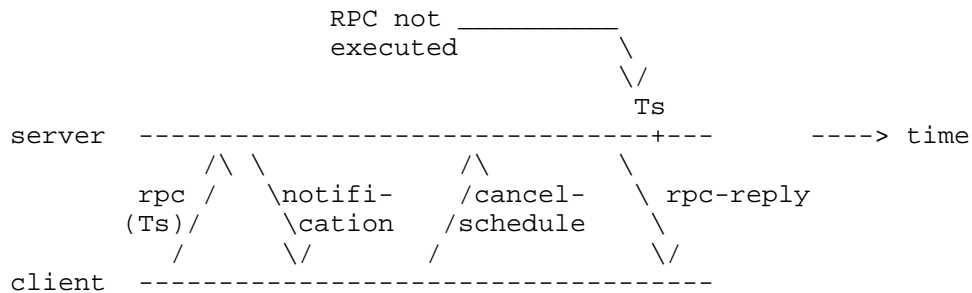


Figure 5 Cancellation Message

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 T_s , the server MUST verify that the value T_s is not too far in the past or in the future. As illustrated in Figure 6, the server verifies that T_s is within the scheduling tolerance range.

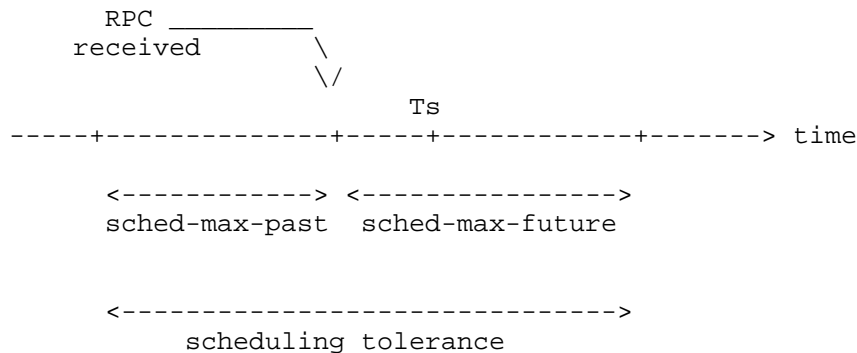


Figure 6 Scheduling Tolerance

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, T_s is within the scheduling tolerance range, the scheduled RPC is performed; if T_s occurs in the past and within the scheduling tolerance, the server performs the RPC as soon as possible, whereas if T_s is a future time, the server performs the RPC at T_s .

If T_s 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 requests 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

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

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

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

This Internet-Draft will expire on September 02, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. EXI Capability	3
3.1. Overview	3
3.2. Dependencies	3
3.3. Capability Identifier	3
3.4. Dynamic Schema-informed Encoding Negotiation	4
3.5. New Mandatory Operations	5
3.5.1. <start-exi>	5
3.5.2. <stop-exi>	8
3.6. New Optional Operations	8
3.6.1. <enable-schema-encoding>	8
3.6.2. <disable-schema-encoding>	9
4. YANG module for <start-exi> and <stop-exi> Operations	9
5. IANA considerations	14
6. Security Considerations	14
7. Acknowledgements	14
8. Normative References	14
Author's Address	14

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]:

- o capability
- o client
- o message
- o protocol operation
- o remote procedure call
- o 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&schemas=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 to 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 it 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 not 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 it 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

Network Working Group
Internet-Draft
Intended status: Experimental
Expires: January 12, 2014

S. Yang
X. Ji
T. Zou
G. Yan
Huawei Technologies
July 11, 2013

NETCONF rpc-error extension
draft-ysc-netconf-rpc-error-extension-00

Abstract

The NETCONF is a machine-machine interface, it is easy to expand. This document will expand the rpc-error message to make multiple language support easily.

Requirements Language

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

Status of This Memo

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

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

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

This Internet-Draft will expire on January 12, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. The definition in NETCONF Protocol	2
3. The solution	3
4. Error-parameters Capability	3
4.1. Overview	3
4.2. Dependencies	4
4.3. Capability Identifier	4
4.4. New Operation	4
4.5. Modifications to Existing Operations	4
4.6. Interactions with Other Capabilities	5
5. Use Case in NMS	5
6. YANG Module for the <error-parameters>	5
7. IANA Considerations	7
8. Security Considerations	8
9. Acknowledgements	8
10. Normative References	8
Authors' Addresses	9

1. Introduction

The Network Configuration Protocol (NETCONF) provides mechanisms to install, manipulate, and delete the configuration of network devices. It uses a RPC-based communication model. NETCONF peer use <rpc> and <rpc-reply> elements to provide transport protocol-independent framing of NETCONF requests and responses. The <rpc> element is used to enclose a NETCONF request sent from the client to the server. The <rpc-reply> message is sent in response to an <rpc> message. The <rpc-error> element is sent in <rpc-reply> messages if an error occurs during the processing of an <rpc> request. The error-message is part of rpc-error information; it contains a string suitable for human display that describes the error condition.

The network device of one producer may be used in many industries and be integrated with many NMS all over the world, each industry or NMS has different custom and demand in the GUI style. So there is a need for the error-message to support multiple language and customization.

2. The definition in NETCONF Protocol

Although NETCONF already support identify the language type by `xml:lang="en"` in the error-message, but it's very difficult for network devices to support multiple languages or customization for error-message, because of storage limitation, complexity on software release, unexpected customization requirement, and so on.

This document describes another solution to resolve this issue by extending the rpc-error with a new capability: error-parameters.

3. The solution

First of all, we classify all languages into two types: common language and local language. English is specified as common language, and all other languages are specified as local language. Each error-message contains 2 parts, static format string and dynamic error parameters, each format string mapping to a unique an error-app-tag. NMS could translate the format string from common language to local language for each error-app-tag, and network devices could return the error parameters in the rpc-reply. So network devices only need support common language in error-message, NMS could support local language for error-message by combining the format string of local language and error-parameters. It's similar to customization, network device only need support default error-message in common language, and NMS could support customization for error-message.

Example:

error-app-tag: 00010001

Error message: MTU value 25000 of interface ethernet1/0/1 is not within range 256..9192

Error-message definition for common language:

Error-parameters:25000, ethernet1/0/1, 256, 9192

Format string: MTU value \$1 of interface \$2 is not within range \$3..\$4

4. Error-parameters Capability

4.1. Overview

The :error-parameters capability indicates that the device supports to carry error parameters which are referred by error-message in the rpc-error. The error-parameters could be used to support local language and customization for error-message by NMS.

4.2. Dependencies

None.

4.3. Capability Identifier

The :error-parameters capability is identified by the following capability string:

```
urn:ietf:params:netconf:capability:error-parameters:1.0
```

4.4. New Operation

None.

4.5. Modifications to Existing Operations

All operation which may cause an rpc-error carrying error-message which refers error parameter.

The rpc-reply will carry error parameters which are referred by the error-message in the rpc-error.

For example:

```
<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <rpc-error>
    <error-type>application</error-type>
    <error-tag>invalid-value</error-tag>
    <error-severity>error</error-severity>
    <error-app-tag>00010001</error-app-tag>
    <error-path xmlns:t="http://example.com/schema/1.2/config">
      /t:top/t:interface[t:name="Ethernet1/0/1"]/t:mtu
    </error-path>
    <error-message xml:lang="en">
      MTU value 25000 of interface ethernet1/0/1 is not within range 256..9192
    </error-message>
    <error-parameters xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-error-parameters">
      <error-parameter>25000</error-parameter>
      <error-parameter>ethernet1/0/1</error-parameter>
      <error-parameter>256</error-parameter>
      <error-parameter>9192</error-parameter>
    </error-parameters>
  </rpc-error>
</rpc-reply>
```

4.6. Interactions with Other Capabilities

None.

5. Use Case in NMS

One example is provided to describe how this solution support local language for error-message in NMS.

Example:

error-app-tag: 00010001

Error message: MTU value 25000 of interface ethernet1/0/1 is not within range 256..9192

Error-message definition for common language:

Error-parameters:25000, ethernet1/0/1, 256, 9192

Format string: MTU value \$1 of interface \$2 is not within range \$3..\$4

Major work in NMS for each error-message:

1. Translation format string to local language:

Format string: La valeur MTU \$1 de l'interface \$2 n'est pas dans la plage de \$3 a \$4.

Remark: The order of error-parameters in the format string of local language may be different with format string of common language.

2. Search the error-message by error-app-tag and combine the error-parameters into the format string in local language to generate the error-message for local language

La valeur MTU 25000 de l'interface ethernet1/0/1 n'est pas dans la plage de 256 a 9192.

6. YANG Module for the <error-parameters>

<CODE BEGINS> file="ietf-netconf-error-parameters@2013-07-11.yang"

```
module ietf-netconf-error-parameters {  
    namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-error-parameters";
```

```
prefix ncep;

import yuma-netconf { prefix nc; }

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

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

  WG List:    <netconf@ietf.org>

  WG Chair: Bert Wijnen
              <bertietf@bwinjen.net>

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

  Editor: Andy Bierman
              <andy.bierman@brocade.com>

  Editor: Balazs Lengyel
              <balazs.lengyel@ericsson.com>";

description
  "This module defines an extension to the NETCONF protocol
  that allows the NETCONF server to return error parameters in
  the rpc-error which are referred in the error-message.

  Copyright (c) 2013 IETF Trust and the persons identified as
  the document authors. All rights reserved.

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

  This 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-ysc-netconf-rpc-error-extension-00.txt
```



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

revision 2013-07-11 {
  description
    "Initial version.";
  reference
    "RFC XXXX: NETCONF rpc-error extension";
}

grouping ErrorParameters {
  description
    "Contains the <error-parameters> for the <rpc-error> extension.";

  container error-parameters {
    description
      "The container of all error parameters in the <rpc-error>";
    reference
      "RFC XXXX; Section 4.5";

    leaf-list error-parameter {
      type string;
      description
        "error-parameter element";
      reference
        "RFC XXXX; Section 4.5";
    }
  }
}

// extending the rpc-error
augment /nc:rpc-reply/nc:rpc-error {
  description
    "Adds the <error-parameters> parameter to the <rpc-error>.";
  reference
    "RFC XXXX; Section 4.5";

  uses ErrorParameters;
}
}
```

<CODE ENDS>

7. IANA Considerations

This document registers the following capability identifier URN in the 'Network Configuration Protocol (NETCONF) Capability URNs' registry:

urn:ietf:params:netconf:capability:error-parameters:1.0

This document registers two XML namespace URNs in the 'IETF XML registry', following the format defined in [RFC3688].

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-error-parameters

Registrant Contact: The NETCONF WG of the IETF.

XML: N/A, the requested URIs are XML namespaces.

This document registers one module name in the 'YANG Module Names' registry, defined in [RFC6020] .

name: ietf-netconf-error-parameters

prefix: ncep

namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-error-parameters

RFC: XXXX

8. Security Considerations

This document does not introduce any further security issues other than that discussed in [RFC6241].

9. Acknowledgements

NA

10. Normative References

- [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., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

Authors' Addresses

Shouchuan Yang
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095
China

Email: yangshouchuan@huawei.com

Xiaofeng Ji
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095
China

Email: jixiaofeng@huawei.com

Ting Zou
Huawei Technologies
Santa Clara-2330 Central Expressway
Santa Clara, CA 95050
America

Email: Tina.Tsou.Zouting@huawei.com

Gang Yan
Huawei Technologies
Huawei Bld., No.156 Beiqing Rd.
Beijing 100095
China

Email: yanggang@huawei.com