

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.

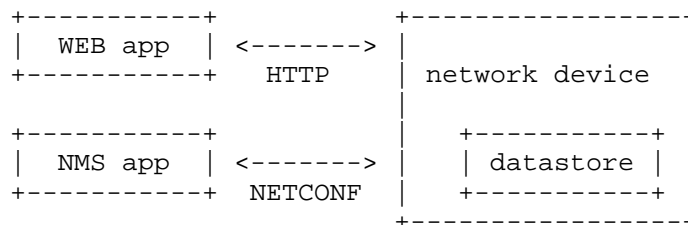
A simplified transaction model is needed that allows basic CRUD operations on a hierarchy of conceptual resources. This represents a limited subset of the transaction capabilities of the NETCONF protocol.

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

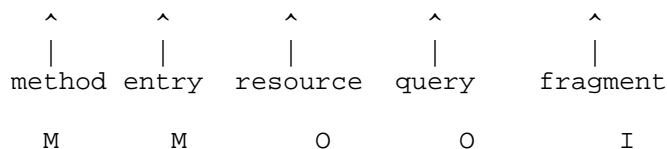
This section describes the messages that are used in the YANG-API protocol.

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

Network Working Group
Internet-Draft
Updates: 6020 (if approved)
Intended status: Standards Track
Expires: April 8, 2013

M. Bjorklund
Tail-f Systems
L. Lhotka
CZ.NIC
October 5, 2012

Operational Data in NETCONF and YANG
draft-bjorklund-netmod-operational-00

Abstract

This document defines the concept of operational state data in the context of YANG and the Network Configuration Protocol (NETCONF). It updates RFC 6020 with rules for how to model the operational state, and defines NETCONF operations to retrieve and modify the operational state.

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 April 8, 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	3
1.1. Terminology	3
1.1.1. Terms	3
2. Objectives	4
3. Problem Statement	5
3.1. Modeling and Retrieving Operational State	5
3.1.1. Example: Interface List	5
3.2. Modifying the Operational State	6
3.2.1. Example: Routing Table Modification	7
4. Datastores	8
4.1. Operational State Datastore	8
4.2. Configuration Datastore	9
5. Constraints	10
5.1. Alternative A	10
5.2. Alternative B	10
5.3. Alternative C	10
6. Protocol Operations	11
6.1. <get-operational>	11
6.1.1. Example: Ethernet Duplex	11
6.2. <edit-operational>	12
7. YANG Module	13
8. IANA Considerations	15
9. Security Considerations	16
10. References	17
10.1. Normative References	17
10.2. Informative References	17
Appendix A. Example: Interface List	18
Appendix B. Example: Ethernet Duplex	19
Appendix C. Example: Admin vs. Oper State	20
Authors' Addresses	21

1. Introduction

1.1. Terminology

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

1.1.1. Terms

The following terms are defined in [RFC6241] and are not redefined here:

- o client
- o configuration datastore
- o datastore
- o server

The following terms are defined in [RFC6020] and are not redefined here:

- o data model
- o schema tree
- o data node

The following terms are used within this document:

- o operational state data: The data in the operational state datastore.
- o operational state datastore: A conceptual data structure from which one can determine device state and behavior.

2. Objectives

- o Develop a general model applicable not only to NETCONF but also to other approaches (RESTful, editable state data etc.).
- o Develop a specific model for NETCONF and YANG.
- o As little changes to NETCONF and YANG as possible.
- o Clarification of the terms "operational state data" and "configuration".

3. Problem Statement

3.1. Modeling and Retrieving Operational State

The NETCONF operation <get> returns both device state data and the running configuration. Quite often, device parameters require a dual representation, both as configuration and state data.

For instance, an IP address may be specified in an interface configuration but, depending on other circumstances, this address may not be used for that interface. In any case, an operator should be able to obtain the addresses that are in operational use.

This implies that some state data must be modeled separately from the configuration data, which leads to a certain amount of duplication in data models. This approach has other drawbacks, too. It is counter-intuitive to data model designers, for whom configuration and state parameters are closely related (see Section 3.1.1 for an example). Data model duplication is error prone and leads to bigger data models, that are more difficult to understand. Further, there is no formal information in the data model about the relationship between the configuration and operational state data.

3.1.1. Example: Interface List

Suppose we want to model a list of interfaces. We allow pre-configuration, i.e., it is legal to configure an interface for which there is currently no hardware present in the system. In this simple example, each interface has a name and a counter of the number of packets received. The counter is operational state data.

```
list interface {
  key name;

  leaf name { ... }
  leaf in-packets {
    type yang:counter64;
    config false;
  }
  ...
}
```

A particular device has hardware for two interfaces with names "eth0" and "eth1". In the configuration there is:

```
<interface>
  <name>eth0</name>
  ...
</interface>
<interface>
  <name>eth2</name>
  ...
</interface>
```

We can see this by doing `<get-config>`.

Operationally, however, the interfaces used are "eth0" and "eth1", although "eth1" does not have any configuration and does not send or receive packets.

How can an operator learn about the presence of "eth1"? The `<get>` operation returns the running configuration and state data together. So, `<get>` will not show "eth1", since it is not present in the running configuration.

With NETCONF as currently defined, the only alternative is to duplicate the data model:

```
list interface {
  key name;

  leaf name { ... }
  ...
}

list interface-oper {
  config false;
  key name;

  leaf name { ... }
  leaf in-packets { ... }
  ...
}
```

3.2. Modifying the Operational State

In some cases, it is useful for clients to directly modify the operational state. An example of this is the recent discussions around an Interface to the Routing System (IRS), where a client needs to modify the routing table, without storing routes in the configuration.

With NETCONF as currently designed, the only way to do this is to

define separate rpc operations. This leads to another kind of data model duplication, where every writable parameter is modeled both as state data that can be retrieved using the <get> operation, and also as input parameters to at least one rpc operation.

3.2.1. Example: Routing Table Modification

Suppose we want to model IPv4 routing tables as operational state, and we also want to be able to let a client modify this data. We have to do:

```
list routing-table {
  config false;
  key name;

  leaf name { ... }
  list route {
    key id;

    leaf id { ... }
    leaf dest-prefix { ... }
    leaf next-hop { ... }
    ...
  }
}

rpc add-route {
  input {
    leaf routing-table-name { ... }
    leaf route-id { ... }
    leaf dest-prefix { ... }
    leaf next-hop { ... }
  }
}

rpc delete-route {
  input {
    leaf routing-table-name { ... }
    leaf route-id { ... }
  }
}
```

4. Datastores

The fundamental idea of this document is to define operational state data as an explicit data structure called the operational state datastore. It is available to all management interfaces, which includes NETCONF but also other interfaces such as SNMP.

The "running" configuration datastore is viewed as a separate overlay data structure whose layout is identical to the subset of the operational state datastore that represents configuration.

4.1. Operational State Datastore

The operational state datastore consists of all parameters that provide information about the instantaneous state of the device and immediately influence the device's behavior.

The operational state datastore is a conceptual data structure. This means that implementations may choose any suitable representation of the datastore, or even generate it dynamically upon request.

Operational state may be modified through one or more management interfaces, or through the operation of network protocols. All such means of accessing and changing the operational state act conceptually on the same data - the operational state datastore. It means, for instance, that any change caused by a network protocol is immediately visible to all management interfaces.

The schema for the operational state datastore is made up of all data nodes defined in YANG modules, specifically both "config true" and "config false" data nodes.

Note that when <get-config> is used to retrieve a "config true" node, the value stored in the configuration datastore is returned. When <get-operational> is used to retrieve the same node, the value actually used by the device is returned. This value may or may not be the same as the value in the datastore.

```
+-----+
| Open Question                                     |
+-----+
| Should there be a YANG statement 'operational <bool>' so that |
| config true nodes can be marked as not being part of the     |
| operational schema?                                           |
+-----+
```

Nodes in the operational data store cannot be directly modified using the standard NETCONF operations.

5. Constraints

This document updates section 8 of RFC 6020 with rules for the operational state datastore.

NOTE: The rest of this section documents some alternatives that the authors want to discuss

There are a couple of design alternatives here:

5.1. Alternative A

No constraints ("must", "mandatory", "unique", "min-elements", "max-elements") are enforced on the operational state datastore. For example, this means that a mandatory "config true" leaf does not have to be present in the operational state datastore.

The problem with this approach is that there is no way to formally define constraints on the OSD in the data model. This may be needed in order to allow for coexistence of NETCONF with other management interfaces that do not use the configuration datastore. Such constraints can be specified in description statement though.

5.2. Alternative B

Change the definitions of mandatory, must, to work on osd instead of config.

This would be a major backwards incompatible change to YANG, and it would not be possible to define constraints on the configuration.

5.3. Alternative C

Introduce new YANG statements for OSD constraints, e.g. osd:must, osd:mandatory etc.

The drawback with this is that it adds complexity.

6. Protocol Operations

6.1. <get-operational>

This document introduces a new operation <get-operational>, which is used to retrieve the operational state data from a device. Note how this operation differs from <get>, which is used to retrieve both the running configuration and state data.

<get-operational> takes the same parameters as <get>.

Since leafs with default values defined in the data model are always explicitly set in the operational data store, there is no need for :with-defaults handling in the <get-operational> operation.

6.1.1. Example: Ethernet Duplex

As an example, consider a very simplified data model with a single leaf for ethernet duplex:

```
leaf duplex {  
  type enumeration {  
    enum "half";  
    enum "full";  
    enum "auto";  
  }  
  config true;  
}
```

Suppose a device with this data model implements the candidate datastore. The following is an example of data from such a device:

get-config from candidate:

```
<duplex>half</duplex>
```

get-config from running:

```
<duplex>auto</duplex>
```

get-operational:

```
<duplex>full</duplex>
```


In this example, the running configuration tells the device to negotiate the duplex mode, and the current, operationally used, value is "full". At the same time, the (uncommitted) candidate configuration contains the value "half".

6.2. <edit-operational>

[Editor's note: NOT FINISHED - not clear if we need this]

Introduce edit-operational. This modifies the subset of the operational data tree that is also marked as writable.

Drawback: does not handle persistent operational data. If we have persistent operational data, this has to be its own data store that can be read and written.

A data model w/o the writable markers cannot be written to. This is a problem, since it is not obvious that the original designer thought about future use cases. For example, our route tables are read-only. Then IRS comes along and wants to write to this data. Do we have to update our spec? Not good. One option is for IRS to publish a deviation data model that added the writable statement to our model. This would be backwards compliant and good. Even better would be if they could publish a conformance statement in a module, w/o the need for deviations.

7. YANG Module

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

<CODE BEGINS> file "ietf-netconf-operational.yang"

```
module ietf-netconf-operational {

    namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-operational";
    prefix "oper";

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

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

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

```
    }  
  }  
}  
  
rpc edit-operational {  
  input {  
    leaf default-operation {  
      type enumeration {  
        enum merge {  
          description  
            "The default operation is merge.";  
        }  
        enum replace {  
          description  
            "The default operation is replace.";  
        }  
        enum none {  
          description  
            "There is no default operation.";  
        }  
      }  
      default "merge";  
      description  
        "The default operation to use.";  
    }  
  }  
  
  choice edit-content {  
    mandatory true;  
    description  
      "The content for the edit operation."  
  
    anyxml data {  
      description  
        "Inline data content."  
    }  
    leaf url {  
      if-feature nc:url;  
      type inet:uri;  
      description  
        "URL-based config content."  
    }  
  }  
}  
}
```

<CODE ENDS>

8. IANA Considerations

This document registers a 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-netconf-operational

Registrant Contact: The NETMOD WG of the IETF.

XML: N/A, the requested URI is an XML namespace.

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

name:	ietf-netconf-operational
namespace:	urn:ietf:params:xml:ns:yang:ietf-netconf-operational
prefix:	oper
reference:	RFC XXXX

9. Security Considerations

This document does not introduce any new security concerns in addition to those specified in [RFC6020] and [RFC6241].

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.
- [RFC6020] Bjorklund, M., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, October 2010.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

10.2. Informative References

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

Appendix A. Example: Interface List

With the proposed solution, the interface list example from ^ex-if-list-2, can be solved with a single list:

```
list interface {
  key name;

  leaf name { ... }
  leaf in-packets {
    type yang:counter64;
    config false;
  }
  ...
}
```

The operation <get-operational> will return the interfaces available on the device:

```
<interface>
  <name>eth0</name>
  ...
</interface>
<interface>
  <name>eth1</name>
  ...
</interface>
```

And <get-config> on running will return the configured interfaces, just as before:

```
<interface>
  <name>eth0</name>
  ...
</interface>
<interface>
  <name>eth2</name>
  ...
</interface>
```

Appendix B. Example: Ethernet Duplex

A typical problem is when the value space for the configuration data is a super set of the value space for the operational state data. An example of this is Ethernet duplex, which can be configured as "half", "full", or "auto", but the operationally used value is either "half" or "full". Without the definition of operational state in this document, this would have to be modeled as two separate leafs:

```
leaf duplex {
  type enumeration {
    enum "half";
    enum "full";
    enum "auto";
  }
}

leaf oper-duplex {
  type enumeration {
    enum "half";
    enum "full";
  }
}
```

With the solution defined in this document, a single leaf is sufficient:

```
leaf duplex {
  type enumeration {
    enum "half";
    enum "full";
    enum "auto";
  }
}
```


Appendix C. Example: Admin vs. Oper State

Another common problem is when the value space for the configured data is a subset of the operational state data. An example is an interface's desired state, and its operational state. The desired state can be "up" or "down", but the operational state can be "up", "lower-layer-down", "testing", etc.

These kind of situations are still best modeled as two separate leafs, one "admin-state" and one "oper-state".

Authors' Addresses

Martin Bjorklund
Tail-f Systems

Email: mbj@tail-f.com

Ladislav Lhotka
CZ.NIC

Email: lhotka@nic.cz

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/>

Conditional Enablement of Configuration Nodes
draft-kwatsen-conditional-enablement-00

Abstract

This memo presents a cross-cutting technique whereby a NETCONF server can support conditional enablement of configuration nodes. That is, whether the node is active or not depends on the evaluation of an expression. Two expression types are defined herein, one for latent configuration (present but not actualized) and another for temporal configuration (actualized based on time). This solution presented is extensible so that additional expression types may be added in the future.

Status of This Memo

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

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

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

This Internet-Draft will expire on August 22, 2013.

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. Requirements Terminology	3
2. Introduction	3
3. Motivation	3
3.1. Explicit Nodes Defined in NETMOD Drafts	3
3.2. Precedent in Juniper's JUNOS-Based Products	4
3.3. Use-Cases Scoped By I2RS Working Group	5
4. Expression Types	5
4.1. Overview	5
4.2. Simple Expressions	5
4.3. Complex Expressions	5
5. Feature Types	6
5.1. Overview	6
5.2. Simple	6
5.3. Time	7
6. Conditional Enablement Capability	7
6.1. Overview	7
6.2. Dependencies	7
6.3. Capability Identifier	7
6.4. New Operations	7
6.5. Modifications to Existing Operations	7
6.6. Interactions with Other Capabilities	8
7. Security Considerations	8
8. IANA Considerations	8
9. Normative References	8

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 memo presents a cross-cutting technique whereby a NETCONF server can support conditional enablement of configuration nodes. That is, whether the node is active or not depends on the evaluation of an expression. Two expression types are defined herein, one for latent configuration (present but not actualized) and another for temporal configuration (actualized based on time). This solution presented is extensible so that additional expression types may be added in the future.

3. Motivation

3.1. Explicit Nodes Defined in NETMOD Drafts

Two separate drafts presented during the NETMOD meeting and IETF 85 had data-models with explicit "enabled" leaves (see examples below). One of the questions asked during the sessions was if cross-cutting concerns, such as if a node were enabled, wouldn't be better supported via meta-data, to which there was general agreement.

Example of an "enabled" leaf from
draft-ietf-netmod-routing-cfg-06.txt:

```
leaf enabled {
    type boolean;
    default "true";
    description
        "Enable/disable the router instance.

        If this parameter is false, the parent router instance is
        disabled, despite any other configuration that might be
        present."
}
```

Figure 1

Example of an "enabled" leaf from
draft-ietf-netmod-system-mgmt-04.txt:

```
leaf enabled {
    type boolean;
    default true;
    description
        "Indicates whether this server is enabled for use or not.";
}
```

Figure 2

3.2. Precedent in Juniper's JUNOS-Based Products

Further, there is already a precedent for this strategy in Juniper's JUNOS-based products, where any configuration node can be flagged with an XML attribute stating that it is inactive.

Example of a JUNOS "inactive" statement:

```
<interface inactive="inactive">...</interface>
```

Figure 3

Additionally, JUNOS also supports the notion of a list of conditionals that must all be satisfied for an associated configuration to be applied. JUNOS supports the conditionals to be based on chassis type, model type, routing engine, member, and time.

Example of a JUNOS "when" statement (this is not YANG):

```
groups {
    my-group-g1 {
        system {
            hostname xyz;
        }
        when {
            model tx1000;
            routing-engine re0;
            time 2am to 4am;
        }
    }
}
```

Figure 4

3.3. Use-Cases Scoped By I2RS Working Group

Lastly, discussions with the I2RS Working Group have revealed that they believe they have a need for a device to autonomously switch configuration settings based on time.

4. Expression Types

4.1. Overview

Conditional expressions are boolean expressions that evaluate to either "true" or "false".

Expressions are contained inside an XML attribute called "enabled". An expression that evaluates to "true" enables the associated node, whereas an expression that evaluates to "false" disables it.

A configuration node having no expression set is equivalent to an expression evaluating to "true". That is, all nodes are enabled by default.

Example usage:

```
<foobar enabled="<expression>"/>
```

4.2. Simple Expressions

Simple expressions are the most trivial expressions possible; they are simply either the constant value "true" or "false".

```
expressions = "true" / "false";
```

Simple expressions are useful to disable a configuration node until it is explicitly reenabled. Since this is such a common use-case, this draft enables simple expressions to be supported without having to implement support for complex expressions.

A device advertises support for simple expressions using the feature "simple" in its capability string:

```
<capability string>?features=simple
```

4.3. Complex Expressions

All expressions that are not "simple" are "complex". These expressions require being able to compare values and evaluate logical expressions; they do not need to perform arithmetic, bitwise operations, or assignments.

The grammar is the same for all complex expressions, the only thing that varies is what variable assignments the device supports (e.g. time, reference to a statistical value, etc.).

A device does not explicitly advertise support for complex expressions. Support for complex expressions are implicit when any feature depending on complex expressions (e.g. time) is advertised. For instance:

```
<capability string>?features=time
```

Complex expressions use the following grammar:

```
<expression>  = [ <paren-expr> / <and-expr> / <or-expr> /
                  <not-expr> / <static-expr> / <simple-expr> ]
<paren-expr>  = "(" <expression> ")"
<and-expr>    = <expression> " && " <expression>
<or-expr>     = <expression> " || " <expression>
<not-expr>    = " ! " <expression>
<static-expr> = "true" / "false"
<simple-expr>  = <variable> <operator> <value>
<variable>    = *char
<operator>    = " == " / " != " / " > " / " < " / " <= " / " >= "
<value>       = *char
```

NOTES

#

Valid <variables> values dependent on what the device advertises
 # support for. # Similarly, valid <value> values are dependent on
 # the <variable> used in the expression. For instance, the "time"
 # feature defines the variables "dayofweek" and "houe", each of which
 # can only be compared to specific values. For instance: ((dayofweek
 # >= "Mon" && <= "Fri) && (hour >= 9am && hour <= 5pm))

5. Feature Types

5.1. Overview

The types of expressions a device supports is advertised as a list of "features" in the capability identifier string.

5.2. Simple

The "simple" feature defines support for constant boolean values "true" and "false". Simple expressions are useful to disable a node until it is explicitly reenabled.

5.3. Time

The "time" feature defines support for complex expressions using time-oriented values such as "dayofweek" and "hour". Time expressions are useful to implement policies that depend on time.

6. Conditional Enablement Capability

6.1. Overview

The :conditional-enablement capability advertises that the NETCONF server supports the ability for nodes in its to be annotated with metadata specifying conditions when it is enabled or its values vary.

6.2. Dependencies

None.

6.3. Capability Identifier

The :conditional-enablement capability is identified by the following capability string:

```
urn:ietf:params:netconf:capability:conditional-enablement:1.0? \
features={name,...}
```

The :conditional-enablement capability URI MUST contain a "features" argument assigned a comma-separated list of names indicating which expression-types the NETCONF peer supports. For example:

```
urn:ietf:params:netconf:capability:conditional-enablement:1.0? \
features=simple,time
```

6.4. New Operations

None.

6.5. Modifications to Existing Operations

The :conditional-enablement capability modifies any operation that transmits configuration, including:

```
<get-config>
<edit-config>
<copy-config>
```

A NETCONF server advertising the :conditional capability indicates that any node in its configuration MAY contain XML attributes defined in the "Overview" section of this capability, even though those attributes were not explicitly defined in its YANG module.

6.6. Interactions with Other Capabilities

None.

7. Security Considerations

There are no known security considerations at this time.

8. IANA Considerations

There are no IANA directives.

9. Normative References

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

Author's Address

Kent Watsen
Juniper Networks

EMail: kwatsen@juniper.net

Network Working Group
Internet-Draft
Intended status: Informational
Expires: August 30, 2013

Y. Teramoto
Kyoto University
R. Atarashi
IIJ Research Laboratory
Y. Atarashi
Alaxala Networks Corp.
Y. Okabe
Kyoto University
Feb 26, 2013

Experience of Designing Network Management System
draft-teramoto-experience-network-management-01

Abstract

This document describes our experiences from designing and implementing a large-scale local area network management system using mainly NETCONF. We designed the data models for device configurations and implemented NETCONF client to centrally control multiple devices of various vendors. The document provides insight on strong and weak points of current NETCONF approach. The document also makes some recommendations about NETCONF and future network management protocols.

Status of this Memo

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

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

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

This Internet-Draft will expire on August 30, 2013.

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	3
2. Management System Setup	3
2.1. Required Information	3
3. Experiences of Implementing NETCONF Client	4
3.1. Transport Protocol	4
3.2. Framing Mechanism on SSH transport	5
3.3. Capability Exchange	5
3.4. Lock and Commit Mechanism	6
3.5. Notification	6
4. Experiences with Data Model Design	7
4.1. YANG	7
5. Experiences with Management Network	7
5.1. VLAN Configuration	7
6. Conclusions and Recommendations	8
7. Security Considerations	9
8. IANA Considerations	9
9. Normative References	9
Authors' Addresses	9

1. Introduction

We designed a large-scale local area network management system that can manage the network independently of physical topology and the vendor of networking equipments composing the network system by modeling the functions of network equipments and using widely used network management protocols. To manage networks, we used mainly the NETCONF protocol and the SNMP [RFC3416] because these protocols are supported by various major vendors' networking equipments and does not depend on specific architectures such as an OpenFlow or a MPLS. This document describes the experiences from designing and implementing such management system.

An NETCONF protocol is defined in [RFC6241] and is intended to manage configurations of the networking equipment from computers. The NETCONF protocol is supported by various network equipments such as Cisco IOS and Junos and so on. We mainly used a NETCONF protocol to get and edit configurations of networking equipments. Although many networking equipments support the NETCONF protocol, there are some kinds of data model designed by vendors. It is because the model layer of NETCONF protocol is still developing. We designed the unified data model for device configurations to manage multiple equipments of various vendors and implemented NETCONF client to control multiple devices simultaneously.

This document evaluates the NETCONF protocol design and makes some recommendations about NETCONF and future network management protocols to support large-scale network management.

2. Management System Setup

Our Network management system contains some information in database to grasp network states and authorize management interface of equipments. The NETCONF protocol and SNMP requires login authorization on starting login. Moreover these protocols support only the information for the configurations and states, and hence the management system needs some information related to several networking equipments such as the network topology and the routes of IP packets. The remainder of this section describes the setup information that is required to start network managing.

2.1. Required Information

Network Management systems usually require some information in advance that the systems cannot retrieve from network devices automatically like followings:

Management Interface

One of the most fundamental information is login authorization for connecting to management interfaces of network devices: management IP address, user ID, password and so on. This is because management protocols such as NETCONF and SNMP usually requires authentication to get or edit device configurations at need to be secured.

Device Information

The system requires the information for devices, such as the vendor, the model and the version to determine the capabilities and providing resource of the device, because NETCONF provides no information and no unified models for them.

Network Information

Network Devices provides no information for network topology. There are some protocols to get neighbor information such as LLDP, however there is no standardized protocols to provide them into client. Therefore, to manage network with network topology, the topology information is to be prepared in advance.

3. Experiences of Implementing NETCONF Client

The NETCONF protocol is an XML-based protocol used to manage the configuration of networking equipment. We implemented NETCONF client to manage networking equipments centrally. This section provides insight on NETCONF protocol.

3.1. Transport Protocol

The NETCONF protocol defines an SSH protocol as mandatory transport protocol. The advantage of using the SSH protocol is that it provides strong authentication mechanism and full encrypted communication. However, there are some issues of using the SSH protocol. It is very large protocol for developers to implement full scratch client, therefore they need to use some existing libraries. Network management systems often have their own architecture for increasing performance, because it is often required high-performance operation and response. SSH libraries also often have their own architecture, and hence it is need to carefully bond two architecture. This may cause performance degradation on multi-threading software because these asynchronous events cannot be concentrated into one. Furthermore, if encrypted communication is forced like the SSH protocol, it is sometimes difficult to detect the

cause of transport problems on debugging.

3.2. Framing Mechanism on SSH transport

The current framing mechanism of SSH transport is defined in [RFC6242]. The previous version of NETCONF (version 1.0) defined framing mechanism to separate each messages by the character sequence "]]>]]>" according to the assumption that well-formed XML documents does not contain the sequence, however it was found later that the assumption is not collect. Therefore the current framing mechanism of SSH uses chunked framing mechanism.

However framing mechanism still have confusing specification. The framing mechanism defines no error notification mechanism when given chunk-size is invalid or an error occurs on transport layer. [RFC6242] requires the peer to terminate the NETCONF session immediately without notifying error information when receiving such an invalid message. This mechanism often causes confusing issues that the developer cannot determine the reason of unexpected disconnection, because the reason may exist on multiple layers from physical layer to application layer. This problem also occurs on the previous version.

3.3. Capability Exchange

Capabilities are advertised in messages sent by each peer during session establishment as <hello> message. Each peer determines the NETCONF version by comparing the sent capabilities with received one. Furthermore this list can contain other additional capabilities such as the capability of notification.

The mechanism of capability exchange have the same undesirable specification as framing mechanism of the SSH transport. [RFC6241] require peers to disconnect the session immediately when the supporting capability version is mismatched or the treatment of session-id is wrong.

The reaction of invalid <hello> varies by vendor implementations. For example, the Junos returns error message as XML comments like followings:

```
<!-- netconf error: unknown command -->
<!-- session end at 2012-08-30 19:01:10 UTC -->
```

Cisco IOS returns no error message and disconnected on receiving next message by the frame chunk. This causes unexpected disconnection on sending first <rpc> command.

This is notable that other management protocols that have capability

exchange such as OpenFlow often support error notification mechanism on receiving an invalid hello message.

3.4. Lock and Commit Mechanism

Lock mechanism of NETCONF is defined in [RFC6242] as mandatory function. Commit mechanism is defined as optional function. These commands are useful on controlling multiple devices.

3.5. Notification

The original NETCONF protocols does not provide networking equipments to push some information to client. To send some information from servers, notification mechanism is defined as optional capability in [RFC5277].

Notification mechanism supports replaying the previous logs by specifying startTime. After starting notification, the server can only operate close-session command to terminate current session and returns resource-denied error on receiving other commands. However, the server that supports interleave capability can operate any commands while notification phase. The client can specify stopTime to stop the notification on the time. After stop time, NETCONF session becomes ordinal mode and accept any NETCONF commands again.

This too complicated mechanism makes enormous number of session states and conditions like following diagram; this makes it difficult to create brief management system that support full NETCONF specifications.

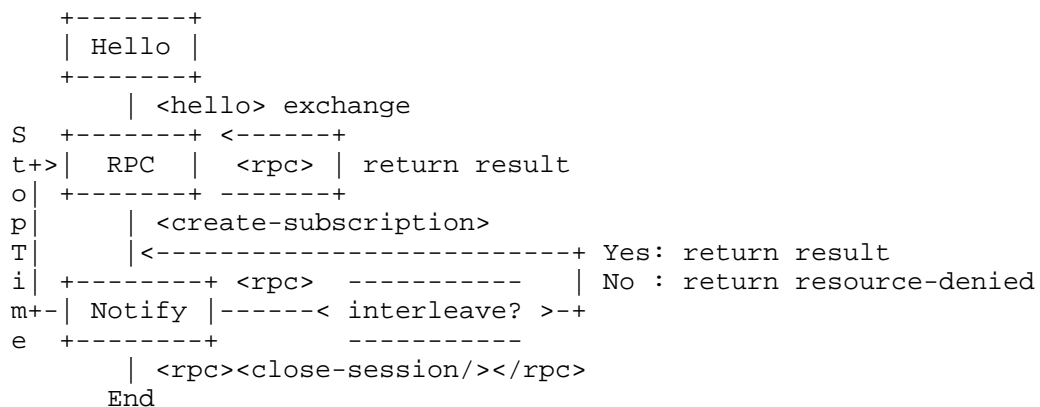


Figure 1: Session States and Conditions

4. Experiences with Data Model Design

Current NETCONF protocol does not provide data models specification, and hence there is no unified data models. Therefore, to manage networking equipments of multiple vendors with same way, we designed some models of the function that networking equipments provides. This section describes the insight on current approach of standardizing data models.

4.1. YANG

YANG is a data modeling language used to model configuration and state data manipulated by the NETCONF defined in [RFC6020]. The approach of YANG is to express various configurations using highly expressive schema, while the approach of SNMP is to predefine fundamental data model.

The same approaches of YANG often causes variability of configuration model and also causes too large and complex data schema. Furthermore YANG defines only the model schema, therefore the way to construct model data is owed to the developer. We think the fundamental model data, that is frequently used in configuration such as VLANs and Interfaces, should be defined by NETCONF core specification like SNMP in order to avoid such problems.

5. Experiences with Management Network

We designed the system to manage multiple network equipments of multiple vendors that support the NETCONF protocol. This section describes the experience with example of designing some function that uses multiple networking equipments via NETCONF.

5.1. VLAN Configuration

VLAN configuration is one of most fundamental functions; however the configuration needs various information. VLAN configuration requires following information:

- o Current VLAN assignment
- o VLAN ID and port name to be assigned
- o Additional information such as bandwidth, port state (some ports may be disabled)
- o Network topology

First two information can be retrieved from NETCONF configurations. The rest information, however, is more difficult than the former two. NETCONF provides no information for the capability of networking equipments, such as ports speed or the number of ports, that is not shown as configuration text. NETCONF also does not offer topology or neighbor information because of the same reason.

No current widely used protocols support the mechanism to retrieve such information, therefore we have no choice but prepare them in advance. However, an OpenFlow protocol now provide the way to get them by handling LLDP packets using packet-out operation. This is one of the reasons that an OpenFlow protocol is referred as a hot protocol.

6. Conclusions and Recommendations

We come to deeply know about the NETCONF protocol on the view of developers. This section conclude our experience and recommends some requirements that future management protocol would satisfy.

The current NETCONF protocol have some undesirable specifications like followings:

- o Specify an SSH protocol as an mandatory transport protocol.
- o Some undesirable error handling such as on capability exchange failure
- o Too many state and conditions of each session by notification

Furthermore, a current approach of models is going toward complicated and large models by YANG. We propose future management protocols to use simple transport protocol, to define appropriate error handling that does not disconnect without any error notification, and to provide simple notification mechanism that is supported by default and supports interleave capability like function as default.

Current Network Management is required to use some similar protocols such as NETCONF and SNMP. We found that the two protocol is covering each other weakness that NETCONF does not support live information and SNMP does not support configuration management. However, there are few ways to get the capability of the networking equipments and the information for constructing topology graphs. There are often required to prepared in advance. There is a need to new management protocols for flexibly control whole network by providing appropriate models and information of the network.

7. Security Considerations

In our experience, the NETCONF protocol require high security considerations on the specification such as authorization and encrypted messaging; therefore the use of the NETCONF protocol improves the security of management.

To manage networking equipments centarally does not matter security issues if they are used in separated logical network and operated proper properly.

8. IANA Considerations

This document makes no request of IANA.

9. Normative References

- [RFC3416] Presuhn, R., "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, December 2002.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, July 2008.
- [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.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.

Authors' Addresses

Yasuhiro Teramoto
Graduate School of Informatics Kyoto University
Yoshida-Hommachi
Sakyo-ku, Kyoto 606-8501
Japan

Phone: +81-75-753-7417
Fax: +81-75-753-7440
Email: teramoto@net.ist.i.kyoto-u.ac.jp

Ray S. Atarashi
IIJ Research Laboratory
Jinbocho Mitsui Bldg., 1-105 Kanda Jinbo-cho
Chiyoda-ku, Tokyo 101-0051
Japan

Phone: +81-3-5205-6464
Fax: +81-3-5205-6466
Email: ray@iijlab.net

Yoshifumi Atarashi
Alaxala Networks Corp.
Shin-Kawasaki Mitsui Bldg.
890 Saiwai-ku Kashimada
Kawasaki, Kanagawa 212-0058
Japan

Phone: +81-44-549-1735
Fax: +81-44-549-1272
Email: atarashi@alaxala.net

Yasuo Okabe
Academic Center for Computing and Media Studies Kyoto-University
Yoshida-Hommachi
Sakyo-ku, Kyoto 606-8501
Japan

Phone: +81-44-549-1735
Fax: +81-44-549-1272
Email: okabe@i.kyoto-u.ac.jp

