

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: October 10, 2013

A. Bierman  
YumaWorks, Inc.  
April 8, 2013

The NETCONF <get2> Operation  
draft-bierman-netconf-get2-03

Abstract

This document describes NETCONF protocol enhancements to improve data retrieval capabilities.

Status of this Memo

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

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

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

This Internet-Draft will expire on October 10, 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
1.1. Problem Statement . . . . .	3
1.1.1. Cannot Retrieve Just the Non-Configuration Data . . . . .	3
1.1.2. No Last-Modified Indication or Time Filtering . . . . .	3
1.1.3. No Simple Instance Discovery Mechanism . . . . .	3
1.1.4. No Subtree Depth Control . . . . .	4
1.1.5. Content Filter Specification is not Extensible . . . . .	4
1.2. Solution . . . . .	4
1.3. Terminology . . . . .	4
1.3.1. NETCONF . . . . .	4
1.3.2. YANG . . . . .	5
1.3.3. Terms . . . . .	5
2. <get2> Operation . . . . .	7
2.1. Depth Filters . . . . .	7
2.2. Time Filters . . . . .	8
3. XSD for the "last-modified" Attribute . . . . .	9
4. <get2> YANG Module . . . . .	10
5. IANA Considerations . . . . .	16
6. Security Considerations . . . . .	17
7. Change Log . . . . .	18
7.1. 00-01 . . . . .	18
7.2. 01-02 . . . . .	18
7.3. 02-03 . . . . .	18
8. References . . . . .	19
8.1. Normative References . . . . .	19
8.2. Informative References . . . . .	19
Appendix A. Examples . . . . .	20
A.1. YANG Module Used in Examples . . . . .	20
A.2. YANG Data Used in Examples . . . . .	21
A.3. Example: If-Modified-Since Non-Empty Filter Retrieval . . . . .	21
A.4. Example: If-Modified-Since Empty Filter Retrieval . . . . .	22
A.5. Example: Keys Only Filter Retrieval . . . . .	23
A.6. Example: Testing for Node Existence with Depth=1 . . . . .	24
A.7. Example: Keys Only Filter Retrieval with Depth=3 . . . . .	25
A.8. Example: Retrieve Only Non-Configuration Data Nodes . . . . .	26
Author's Address . . . . .	28

## 1. Introduction

There is a need for standard mechanisms to allow NETCONF [RFC6241] application designers to retrieve data from NETCONF servers more efficiently.

### 1.1. Problem Statement

This document attempts to address the following problems with NETCONF data retrieval mechanisms.

#### 1.1.1. Cannot Retrieve Just the Non-Configuration Data

The NETCONF <get> operation allows a client to retrieve data from the server but it returns all data, including configuration datastore nodes. The <get-config> operation already returns all configuration datastore nodes.

It was originally thought that <get> should return all nodes so the client would not have to correlate configuration and non-configuration data nodes, since they would be mixed together in the reply.

Operational experience has shown that the <get> operation without reasonable filters to reduce the returned data can significantly degrade device performance and return enormous XML instance documents in the <rpc-reply>.

#### 1.1.2. No Last-Modified Indication or Time Filtering

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

#### 1.1.3. No Simple Instance Discovery Mechanism

Sometimes the client application wants to discover what data exists on the server, particularly list entries. There is a need for a simple mechanism to retrieve just the key leaf nodes within a subtree.

The NETCONF subtree filtering mechanism does provide a very complex way for the client to request just key leafs for specific list entries. A simpler mechanism is needed which will allow the client to discover the list instances present.

#### 1.1.4. No Subtree Depth Control

NETCONF filters allow the client to select specific sub-trees within the conceptual datastore on the server. However, sometimes the client does not really need the entire subtree, which may contain many nested list entries, and be very large.

There is sometimes a need to limit the depth of the sub-trees retrieved from the server. A consistent and simple algorithm for determining what data nodes start a new level is needed.

#### 1.1.5. Content Filter Specification is not Extensible

The NETCONF <get> and <get-config> operations use a hard-coded content filtering mechanism. They use a "type" XML attribute to indicate which of two filter specification types they support, and a "select" XML attribute if the :xpath capability is supported and an XPath [XPATH] expression filter specification is provided.

This design does not allow additional content filter specification types to be supported by an implementation. It does not allow the standard to be easily extended in a modular fashion.

In addition, this design does not allow YANG statements to be used to properly describe the protocol operation. The special "get-filter-element-attributes" YANG extension in the ietf-netconf module is not extensible, and it does not really count as proper YANG, since this extension is outside the YANG language definition.

### 1.2. Solution

This document defines a new NETCONF protocol operation called <get2> to address the deficiencies described in the previous section. It can be implemented existing NETCONF servers without requiring a change in the protocol version.

### 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

#### 1.3.2. YANG

The following terms are defined in [RFC6020]:

- o anyxml
- o container
- o data node
- o key leaf
- o leaf
- o leaf-list
- o list
- o presence container (or P-container)
- o non-presence container (or NP-container)

#### 1.3.3. Terms

The following terms are defined:

- o depth filter: A mechanism implemented within the NETCONF server to allow a client to retrieve only a limited number of levels within the a subtree, instead of retrieving the entire subtree.

- o time filter: A mechanism implemented within the NETCONF server to allow a client to retrieve only data that has been modified since a specified data and time.

## 2. <get2> Operation

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

The <get2> protocol operation contains the following input parameters:

- o source: A container indicating the conceptual datastore for the retrieval request.
- o filter-spec: A choice indicating the content filter specification for the retrieval request.
- o keys-only: A leaf indicating that only the key leaves, combined with other filtering criteria, should be returned.
- o if-modified-since: A leaf indicating the time filter specification for the retrieval request, according to the procedures in Section 2.2.
- o depth: A leaf indicating the subtree depth level for the retrieval request, according to the procedures in Section 2.1.
- o with-defaults: A leaf indicating the type of defaults handling requested, according to procedures in [RFC6243].
- o with-timestamps: A leaf indicating that "last-modified" XML attributes are requested, encoded according to the schema in Section 3.

### 2.1. Depth Filters

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

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

If no content filters are provided, then level 1 is considered to

include all top-level data nodes within the source datastore. Otherwise only the levels in selected subtrees will be considered, and not any additional top-level data nodes.

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

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

## 2.2. Time Filters

A time filter indicates that only data which has been modified since the indicated date and time should be included in the reply.

If this feature is supported, then the server will maintain a last-modified timestamp for the source datastore. It MAY support additional nested timestamps for data nodes within the datastore.

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

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



### 3. XSD for the "last-modified" Attribute

The following XML Schema document [XSD] defines the "last-modified" attribute, described within this document. This XSD is only relevant if the server supports the "timestamps" YANG feature within the "ietf-netconf-get2" YANG module.

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

<CODE BEGINS> file "last-modified.xsd"

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

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

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

<CODE ENDS>
```

#### 4. <get2> YANG Module

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

Several YANG features are imported from [RFC6241].

Some data types are imported from [RFC6021].

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

<CODE BEGINS> file "ietf-netconf-get2@2013-04-08.yang"

```
module ietf-netconf-get2 {

  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-get2";
  prefix get2;

  import ietf-inet-types {
    prefix inet;
  }

  import ietf-netconf {
    prefix nc;
  }

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

  import ietf-yang-types {
    prefix yang;
  }

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

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

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

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

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

description

"This module contains a collection of YANG definitions for the retrieval of information from a NETCONF server.

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

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

This version of this YANG module is part of RFC XXXX; see the RFC itself for full legal notices.";

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

// RFC Ed.: remove this note  
// Note: extracted from draft-bierman-netconf-get2-03.txt

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

```
revision "2013-04-08" {  
  description  
    "Initial revision.";  
  reference  
    "RFC XXXX: The NETCONF <get2> Operation";  
}
```

/\* Features \*/

```
feature timestamps {  
  description  
    "This feature indicates that the server implements  
    the <get2> operations parameters which require  
    last modification timestamps to be maintained by  
    the server.  
  
    If this feature is advertised then one global  
    'last-modified' timestamp for the entire  
    running datastore MUST be supported.
```

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

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

/* Protocol Operations */

rpc get2 {
  description
    "Retrieve NETCONF datastore information";
  input {
    container source {
      choice datastore-source {
        default running;
        description
          "The configuration source for the retrieval operation.
          The running configuration is the default choice if
          this parameter is not present.";
        leaf candidate {
          if-feature nc:candidate;
          type empty;
          description
            "The candidate configuration datastore is the
            retrieval source.";
        }
        leaf running {
          type empty;
          description
            "The running configuration datastore is the
            retrieval source.";
        }
      }
      leaf startup {
        if-feature nc:startup;
        type empty;
        description
          "The startup configuration datastore is the
```

```
        retrieval source.";
    }
    leaf url {
        if-feature nc:url;
        type inet:uri;
        description
            "The URL-based configuration is the
             retrieval source.";
    }
    leaf nonconfig {
        type empty;
        description
            "The retrieval source is the collection of all
             non-configuration data nodes supported by the server.
             Any ancestor container and/or list and list key nodes
             are also returned. No other leafs or leaf-lists will
             be included in the reply.

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

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

leaf keys-only {
    type empty;
    description
```

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

  leaf if-modified-since {
    if-feature timestamps;
    type yang:date-and-time;
    description
      "This parameter selects only data nodes which
        have been modified since the specified time.";
  }

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

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

        If this parameter is greater than '0', then
        only the specified number of subtree levels will
        be returned.";
    reference "RFC XXXX, section 2.1.";
  }

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

  leaf with-timestamps {
    if-feature timestamps;
    type empty;
    description
      "This parameter will cause the server to return
        XML attributes identifying the last modification
        time within one or more elements within the
        <rpc-reply>.";
  }
```

```
        reference "RFC XXXX, sections 2.2 and 3.";
    }

}

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

}

<CODE ENDS>
```

## 5. IANA Considerations

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

URI: urn:ietf:params:xml:ns:yang:ietf-netconf-get2  
Registrant Contact: The NETCONF WG of the IETF.  
XML: N/A, the requested URI is an XML namespace.

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

name: ietf-netconf-get2  
namespace: urn:ietf:params:xml:ns:yang:ietf-netconf-get2  
prefix: get2  
reference: RFC XXXX



## 6. Security Considerations

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

## 7. Change Log

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

### 7.1. 00-01

- o removed subtree-filter YANG feature
- o changed depth filter to exactly match the XML layering
- o renamed filter to subtree-filter
- o renamed select to xpath-filter
- o added some new examples

### 7.2. 01-02

- o added operational data source support
- o added 'ietf-netconf-data-source' module
- o clarified terminology

### 7.3. 02-03

- o removed operational data source support since this problem needs to be solved as part of the operational state work.
- o removed 'operational' datastore from datastore-choice and replaced it with a datastore value called 'nonconfig'
- o removed ietf-netconf-data-source YANG module

## 8. References

### 8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC6020] Bjorklund, M., "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.
- [RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, June 2011.
- [XPath] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation REC-xpath-19991116, November 1999, <<http://www.w3.org/TR/1999/REC-xpath-19991116>>.
- [XSD] Malhotra, A. and P. Biron, "XML Schema Part 2: Datatypes Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-2-20041028, October 2004, <<http://www.w3.org/TR/2004/REC-xmlschema-2-20041028>>.

### 8.2. Informative References

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

## Appendix A. Examples

## A.1. YANG Module Used in Examples

```
module example-get2 {  
  
    namespace "http://example.com/ns/example-get2";  
    prefix exget2;  
    description "Module used in <get2> examples.";   
    revision 2013-04-08;  
  
    container forests {  
        list forest {  
            key name;  
  
            leaf name {  
                type string;  
            }  
  
            leaf tree-count {  
                config false;  
                type uint32;  
            }  
  
            container trees {  
                list tree {  
                    key name;  
  
                    leaf name {  
                        type string;  
                    }  
                    leaf location {  
                        type string;  
                    }  
                    leaf height {  
                        config false;  
                        type decimal64 {  
                            fraction-digits 3;  
                        }  
                        units meters;  
                    }  
                } // list tree  
            } // container trees  
        } // list forest  
    } // container forests  
}
```

## A.2. YANG Data Used in Examples

The follow instances are assumed in the following examples.

```
list forest: "north":  
  list tree: "birch", "ash", "maple"  
  
list forest: "south":  
  list tree: "banyan", "palm"
```

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

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

## A.3. Example: If-Modified-Since Non-Empty Filter Retrieval

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

- o The forest named "north" was last modified after the specified "if-modified-since" timestamp.
- o The forest named "south" was last modified before the specified "if-modified-since" timestamp.
- o The server maintains a last-modified timestamp for the running datastore and the "forest" list entries.
- o The client is also requesting that timestamps be returned for the nodes that have been modified. If any part of the "forest" subtree is modified then this timestamp will be updated.

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-get2">
    <subtree-filter>
      <forests xmlns="http://example.com/ns/example-get2" />
    </subtree-filter>
    <if-modified-since>2012-09-09T01:43:27Z</if-modified-since>
    <with-timestamps />
  </get2>
</rpc>

<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:lm="urn:ietf:params:xml:ns:netconf:last-modified:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-get2"
    lm:last-modified="2012-09-09T02:00:00Z">
    <forests xmlns="http://example.com/ns/example-get2">
      <forest lm:last-modified="2012-09-09T02:00:00Z">
        <name>north</name>
        <trees>
          <tree>
            <name>birch</name>
            <location>hillside</location>
          </tree>
          <tree>
            <name>ash</name>
            <location>southwest pasture</location>
          </tree>
          <tree>
            <name>maple</name>
            <location>east meadow</location>
          </tree>
        </trees>
      </forest>
    </forests>
  </data>
</rpc-reply>
```

#### A.4. Example: If-Modified-Since Empty Filter Retrieval

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

- o No "forest" list entry has been modified since this time so an empty data node is returned.
- o Note that the "last-modified" timestamp is returned for the node representing the datastore, even though no data nodes have been

modified since the specified time. This allows the client to easily retrieve the last-modified timestamp for the entire datastore.

```
<rpc message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-get2">
    <subtree-filter>
      <forests xmlns="http://example.com/ns/example-get2" />
    </subtree-filter>
    <if-modified-since>2012-09-09T03:43:27Z</if-modified-since>
    <with-timestamps />
  </get2>
</rpc>

<rpc-reply message-id="102"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  xmlns:lm="urn:ietf:params:xml:ns:netconf:last-modified:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-get2"
    lm:last-modified="2012-09-09T02:00:00Z" />
</rpc-reply>
```

#### A.5. Example: Keys Only Filter Retrieval

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

- o The default source (running) is used.
- o The default depth="0" is used to retrieve all subtree levels.
- o The "keys-only" leaf is set
- o The "forests" subtree is selected. The xpath-filter is used instead of the subtree-filter.
- o Whitespace added to xpath-filter element for display purposes only.

```
<rpc message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-get2">
    <xpath-filter xmlns:ex=http://example.com/ns/example-get2">
      /ex:forests
    </xpath-filter>
    <keys-only />
  </get2>
</rpc>

<rpc-reply message-id="103"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-get2">
    <forests xmlns="http://example.com/ns/example-get2">
      <forest>
        <name>north</name>
        <trees>
          <tree>
            <name>birch</name>
          </tree>
          <tree>
            <name>ash</name>
          </tree>
          <tree>
            <name>maple</name>
          </tree>
        </trees>
      </forest>
      <forest>
        <name>south</name>
        <trees>
          <tree>
            <name>banyan</name>
          </tree>
          <tree>
            <name>palm</name>
          </tree>
        </trees>
      </forest>
    </forests>
  </data>
</rpc-reply>
```

#### A.6. Example: Testing for Node Existence with Depth=1

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



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

```
<rpc message-id="104"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-get2">
    <subtree-filter>
      <forests xmlns="http://example.com/ns/example-get2">
        <forest>
          <trees />
        </forest>
      </forests>
    </subtree-filter>
    <depth>1</depth>
  </get2>
</rpc>

<rpc-reply message-id="104"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-get2">
    <forests xmlns="http://example.com/ns/example-get2">
      <forest>
        <name>north</name>
        <trees />
      </forest>
      <forest>
        <name>south</name>
        <trees />
      </forest>
    </forests>
  </data>
</rpc-reply>
```

#### A.7. Example: Keys Only Filter Retrieval with Depth=3

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

- o The default source (running) is used.
- o The "keys-only" leaf is set
- o The "forests" subtree is selected
- o The depth parameter is set to "3" to only retrieve the requested layer (forests), its child nodes (forest), and the key leaf nodes from each "forest" entry. Without the "keys-only" parameter, other leafs from the "forest" list would be returned as well.

```
<rpc message-id="105"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-get2">
    <subtree-filter>
      <forests xmlns="http://example.com/ns/example-get2" />
    </subtree-filter>
    <keys-only />
    <depth>3</depth>
  </get2>
</rpc>

<rpc-reply message-id="105"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-get2">
    <forests xmlns="http://example.com/ns/example-get2">
      <forest>
        <name>north</name>
      </forest>
      <forest>
        <name>south</name>
      </forest>
    </forests>
  </data>
</rpc-reply>
```

#### A.8. Example: Retrieve Only Non-Configuration Data Nodes

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

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

```
<rpc message-id="106"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get2 xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-get2">
    <source><nonconfig /></source>
    <subtree-filter>
      <forests xmlns="http://example.com/ns/example-get2" />
    </subtree-filter>
  </get2>
</rpc>

<rpc-reply message-id="106"
  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <data xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-get2">
    <forests xmlns="http://example.com/ns/example-get2">
      <forest>
        <name>north</name>
        <trees>
          <tree>
            <name>birch</name>
            <height>41.013</height>
          </tree>
          <tree>
            <name>ash</name>
            <height>16.523</height>
          </tree>
          <tree>
            <name>maple</name>
            <height>51.204</height>
          </tree>
        </trees>
      </forest>
      <forest>
        <name>south</name>
        <trees>
          <tree>
            <name>banyan</name>
            <height>91.433</height>
          </tree>
          <tree>
            <name>palm</name>
            <height>83.439</height>
          </tree>
        </trees>
      </forest>
    </forests>
  </data>
</rpc-reply>
```

Internet-Draft

<get2>

April 2013

Author's Address

Andy Bierman  
YumaWorks, Inc.

Email: [andy@yumaworks.com](mailto:andy@yumaworks.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](mailto:mbj@tail-f.com)

Ladislav Lhotka  
CZ.NIC

Email: [lhotka@nic.cz](mailto:lhotka@nic.cz)



NETCONF Working Group  
Internet-Draft  
Obsoletes: 5539 (if approved)  
Intended status: Standards Track  
Expires: April 25, 2013

M. Badra  
LIMOS Laboratory  
A. Luchuk  
SNMP Research  
J. Schoenwaelder  
Jacobs University Bremen  
October 22, 2012

NETCONF Over Transport Layer Security (TLS)  
draft-ietf-netconf-rfc5539bis-01

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 to secure NETCONF exchanges. This document 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 April 25, 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. Conventions Used in This Document . . . . .	3
2. NETCONF over TLS . . . . .	3
2.1. Connection Initiation . . . . .	3
2.2. Connection Closure . . . . .	4
3. Endpoint Authentication, Identification and Authorization . .	4
3.1. Server Identity . . . . .	4
3.2. Client Identity . . . . .	5
3.2.1. Deriving NETCONF Usernames From NETCONF Client Certificates . . . . .	5
3.2.2. Deriving NETCONF Usernames From PSK identities . . . .	7
3.2.3. Remote Configuration . . . . .	7
4. Security Considerations . . . . .	14
5. IANA Considerations . . . . .	15
6. Acknowledgements . . . . .	16
7. Contributor's Address . . . . .	16
8. References . . . . .	16
8.1. Normative References . . . . .	16
8.2. Informative References . . . . .	17
Appendix A. Change Log (to be removed by RFC Editor before publication) . . . . .	17
A.1. From draft-ietf-netconf-rfc5539bis-00 to draft-ietf-netconf-rfc5539bis-01 . . . . .	17
A.2. From draft-badra-netconf-rfc5539bis-02 to draft-ietf-netconf-rfc5539bis-00 . . . . .	17
Authors' Addresses . . . . .	17

## 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 "NETCONF over TLS", which includes support for certificate and pre-shared key (PSK)-based authentication and key derivation, utilizing the protected ciphersuite negotiation, mutual authentication, and key management capabilities of the TLS (Transport Layer Security) protocol, described in [RFC5246].

### 1.1. Conventions Used in This Document

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. NETCONF over TLS

Since TLS is application-protocol-independent, NETCONF can operate on top of the TLS protocol transparently. This document defines how NETCONF can be used within a TLS session.

### 2.1. Connection Initiation

The peer acting as the NETCONF client MUST also act as the TLS client. The client actively opens the TLS connection and the server passively listens for the incoming TLS connection on the TCP port 6513. It MUST therefore send the TLS ClientHello message to begin the TLS handshake. Once the TLS handshake has finished, the client and the server MAY begin to exchange NETCONF data. In particular, the client will send complete XML documents to the server containing <rpc> elements, and the server will respond with complete XML documents containing <rpc-reply> elements. The client MAY indicate interest in receiving event notifications from a server by creating a subscription to receive event notifications [RFC5277]. In this case, the server replies to indicate whether the subscription request was successful and, if it was successful, the server begins sending the event notifications to the client as the events occur within the system.

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 [RFC5539] of this document used the same framing sequence defined in [RFC6242], under the assumption that it could not be found in well-formed XML documents. However, this assumption is not correct [RFC6242]. In order to solve this problem, and at the same time be compatible with existing implementations, this document uses the framing protocol defined in [RFC6242] as following:

The <hello> message MUST be followed by the character sequence ]]>]]. Upon reception of the <hello> message, the receiving peer's TLS Transport layer conceptually passes the <hello> message to the Messages layer. 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.

Implementation of the protocol specified in this document MAY implement any TLS cipher suite that provides mutual authentication [RFC5246].

Implementations MUST support TLS 1.2 [RFC5246] and are REQUIRED to support the mandatory-to-implement cipher suite, which is TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA. This document is assumed to apply to future versions of TLS; in which case, the mandatory-to-implement cipher suite for the implemented version MUST be supported.

## 2.2. Connection Closure

Exiting NETCONF is accomplished using the <close-session> operation. A NETCONF server will process NETCONF messages from the NETCONF client in the order in which they are received. When the NETCONF server processes a <close-session> operation, the NETCONF server SHALL respond and close the TLS session channel. The NETCONF server MUST NOT process any NETCONF messages received after the <close-session> operation. The TLS session is closed as described in [RFC6242] Section 7.2.1.

## 3. Endpoint Authentication, Identification and Authorization

### 3.1. Server Identity

If the server's presented certificate has passed certification path validation [RFC5280] to a configured trust anchor, the client MUST carefully examine the certificate presented by the server to determine if it meets the client's expectations. Particularly, the

client MUST check its understanding of the server hostname against the server's identity as presented in the server Certificate message, in order to prevent man-in-the-middle attacks.

Matching is performed according to the rules and guidelines defined in [RFC6125].

If the match fails, the client MUST either ask for explicit user confirmation or terminate the connection and indicate the server's identity is suspect.

Additionally, clients MUST verify the binding between the identity of the servers to which they connect and the public keys presented by those servers. Clients SHOULD implement the algorithm in Section 6 of [RFC5280] for general certificate validation, but MAY supplement that algorithm with other validation methods that achieve equivalent levels of verification (such as comparing the server certificate against a local store of already-verified certificates and identity bindings).

If the client has external information as to the expected identity of the server, the hostname check MAY be omitted.

### 3.2. Client Identity

The server MUST verify the identity of the client to ensure that the incoming client request is legitimate before the NETCONF session is started.

The NETCONF protocol [RFC6241] requires that the transport protocol's authentication process MUST result in an authenticated client identity whose permissions are known to the server. The authenticated identity of a client is commonly referred to as the NETCONF username.

The username provided by the TLS implementation will be made available to the NETCONF message layer as the NETCONF username without modification. If the username does not comply to the NETCONF requirements on usernames [RFC6241], i.e., the username is not representable in XML, the TLS session MUST be dropped.

Algorithms for mapping certificates or PSK identities (sent by the client) to NETCONF usernames are described below.

#### 3.2.1. Deriving NETCONF Usernames From NETCONF Client Certificates

The algorithm for deriving NETCONF usernames from TLS certificates is patterned after the algorithm for deriving `tmSecurityNames` from TLS



certificates specified in Transport Layer Security (TLS) Transport Model for the Simple Network Management Protocol (SNMP) [RFC6353]. The NETCONF server MUST implement the algorithms for deriving NETCONF usernames from presented certificates that are documented in the ietf-netconf-tls YANG module, defined in Section 3.2.3. This YANG module lets the NETCONF security administrator configure how the NETCONF server derives NETCONF usernames from presented certificates. It also lets different certificate-to-username derivation algorithms be used for different certificates.

When a NETCONF server accepts a TLS connection from a NETCONF client, the NETCONF server attempts to derive a NETCONF username from the certificate presented by the NETCONF client. If the NETCONF server cannot derive a valid NETCONF username from the client's presented certificate, then the NETCONF server MUST close the TLS connection, and MUST NOT accept NETCONF messages over it. The NETCONF server uses one of the following algorithms to produce a NETCONF username from the certificate presented by the NETCONF client:

- o Map a certificate directly to a specified, pre-configured, NETCONF username;
- o Extract the subjectAltName's rfc822Name from the certificate, then use the extracted rfc822Name as the NETCONF username;
- o Extract the subjectAltName's dnsName from the certificate, then use the extracted dnsName as the NETCONF username;
- o Extract the subjectAltName's ipAddress from the certificate, then use the extracted ipAddress as the NETCONF username;
- o Examine the subjectAltName's rfc822Name, dnsName, and ipAddress fields in a pre-defined order. Return the value from the first subjectAltName field that is examined, defined, and populated with a non-empty value. If no subjectAltName field of a specific type is defined, then the examination skips that field and proceeds to examine the next field type. If a subjectAltName field is defined, but the value is not populated, or is populated by an empty value, then the examination skips that field and proceeds to examine the next field type.

The NETCONF server MUST implement all of these algorithms, and allow the deployer to choose the algorithm used. The cert-map list in the ietf-netconf-tls YANG module specifies how a NETCONF server transforms a certificate into a NETCONF username.

If the fingerprint of locally held copy of a trusted CA certificate is configured in the cert-map list in the ietf-netconf-tls YANG

module, and that CA certificate is used to validate the certificate presented by the client, then the NETCONF server uses that cert-map list entry to produce the NETCONF username. This allows multiple client certificates (all signed by the same trusted CA certificate) to be mapped to a NETCONF username by a single entry in the cert-map list.

### 3.2.2. Deriving NETCONF Usernames From PSK identities

Implementations MAY optionally support TLS Pre-Shared Key (PSK) authentication [RFC4279]. RFC4279 describes pre-shared key ciphersuites for TLS. The description of the psk-maps container in the ietf-netconf-tls YANG module, defined in section 3.2.3, specifies how a NETCONF server transforms a TLS pre-shared key into a NETCONF username.

### 3.2.3. Remote Configuration

The ietf-netconf-tls YANG module defines objects for remotely configuring the mapping of TLS certificates and of PSK Identities to NETCONF usernames.

```
module ietf-netconf-tls {  
  
  namespace "urn:ietf:params:xml:ns:yang:ietf-netconf-tls";  
  
  prefix "nctls";  
  
  import ietf-yang-types {  
    prefix yang;  
  }  
  
  import ietf-netconf-acm {  
    prefix nacm;  
  }  
  
  organization  
    "IETF NETCONF (Network Configuration) Working Group";  
  
  contact  
    "WG Web:  <http://tools.ietf.org/wg/netconf/>  
    WG List:  <mailto:netconf@ietf.org>  
  
    WG Chair: Mehmet Ersue  
              <mailto:mehmet.ersue@nsn.com>
```

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

Editor: Mohamad Badra  
<mailto:mbadra@gmail.com>;

#### description

"This module applies to NETCONF over TLS. It specifies how NETCONF servers transform X.509 certificates presented by clients into NETCONF usernames. It also specifies how NETCONF servers transform pre-shared TLS keys into NETCONF usernames.

The cert-maps container in this YANG module is patterned after parts of the SNMP-TLS-TM-MIB defined in RFC 6353. Much of the description text has been copied directly from the SNMP-TLS-TM-MIB, and modified as necessary.

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.: please update the date to the date of publication

```
revision "2012-02-13" {
  description
    "Initial version";
  reference
    "RFC XXXX: NETCONF over Transport Layer Security (TLS)";
}

feature map-certificates {
  description
    "The map-certificates feature indicates that the server implements
    mapping X.509 certificates to NETCONF user names.";
}

feature map-pre-shared-keys {
```

```
description
  "The map-pre-shared-keys feature indicates that the server
  implements mapping TLS pre-shared keys to NETCONF user names.";
}

typedef tls-fingerprint-type {
  type string {
    pattern '([0-9a-fA-F]){2}(:([0-9a-fA-F]){2})*';
  }
}
description
  "A cryptographic signature (fingerprint) value that can be used to
  uniquely reference other data of potentially arbitrary length.";
}

container netconf-config {

  container tls {

    //
    // Objects related to deriving NETCONF usernames from X.509
    // certificates.
    //

    container cert-maps {
      if-feature map-certificates;
      config true;

      description
        "The cert-maps container is used by a NETCONF server to map the
        NETCONF client's presented X.509 certificate to a NETCONF username.

        On an incoming TLS connection, the client's presented certificate
        MUST either be validated based on an established trust anchor, or
        it MUST directly match a fingerprint in the 'cert-map' list. This
        module does not provide any mechanisms for configuring the
        trust anchors; the transfer of any needed trusted certificates
        for certificate chain validation is expected to occur through an
        out-of-band transfer.

        Once the certificate has been found acceptable (either by
        certificate chain validation or directly matching a fingerprint
        in the cert-map list), the cert-map list is consulted to determine
        the appropriate NETCONF username to associate with the remote
        connection. This is done by considering each cert-map list entry
        in order. The cert-map entry's fingerprint determines whether the
        list entry is a match for the incoming connection:"
```

- 1) If the cert-map list entry's fingerprint value matches that of the presented certificate, then consider the list entry as a successful match.
- 2) If the cert-map 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.

Once a matching cert-map list entry has been found, the NETCONF server uses the map-type list to determine how the NETCONF username associated with the session should be determined. See the map-type leaf's description for details on determining the NETCONF username value. If it is impossible to determine a NETCONF username from the cert-map list entry's data combined with the data presented in the certificate, then additional cert-map list entries MUST be searched looking for another potential match. If a resulting NETCONF username mapped from a given cert-map list entry is not compatible with the needed requirements of a NETCONF username, then it MUST be considered an invalid match and additional cert-map list entries MUST be searched looking for another potential match.

If no matching and valid cert-map list entry can be found, then the NETCONF server MUST close the connection, and MUST NOT accept NETCONF messages over it.

Security administrators are encouraged to make use of certificates with subjectAltName fields that can be used as NETCONF usernames so that a single root CA certificate can allow all child certificate's subjectAltName to map directly to a NETCONF usernames via a 1:1 transformation."

```
list cert-map {
  key "key";
  ordered-by user;
  description
    "A single list entry that specifies a mapping for an incoming
    TLS certificate to a NETCONF username.";

  leaf key {
    type string;
    nacm:default-deny-all;
    description
      "The key associated with the cert-map list.";
  }

  container fingerprint {
```

```
choice algorithm-and-hash {
  mandatory true;
  leaf md5 {
    type tls-fingerprint-type;
  }
  leaf sha1 {
    type tls-fingerprint-type;
  }
  leaf sha224 {
    type tls-fingerprint-type;
  }
  leaf sha256 {
    type tls-fingerprint-type;
  }
  leaf sha384 {
    type tls-fingerprint-type;
  }
  leaf sha512 {
    type tls-fingerprint-type;
  }
}
description
  "Specifies the signature algorithm and cryptographic
  signature (fingerprint) used to identify an X.509
  certificate.

  Implementations of this YANG module MAY, but are not
  required to, implement all of these cryptographic signature
  algorithms. Implementations of this YANG module MUST
  implement at least one of these cryptographic signature
  algorithms.

  The available choices may be extended in the future as
  stronger cryptographic signature algorithms become
  available and are deemed necessary.";

reference
  "RFC 5246: The Transport Layer Security (TLS) Protocol
  Version 1.2; Section 7.4.1.4.1, Signature Algorithms";
} // choice algorithm-and-hash
} // container fingerprint

choice map-type {
  leaf specified {
    type nacm:user-name-type;
    description
      "Directly specifies the NETCONF username to be used for this
      certificate.";
  }
}
```

```
leaf-list from-certificate {
  ordered-by user;
  type enumeration {
    enum rfc822Name {
      description
        "Maps a subjectAltName's rfc822Name to a NETCONF username.
        The local part of the rfc822Name is passed unaltered but
        the domain-part of the name MUST be passed in lowercase.
        This mapping results in a 1:1 correspondence between
        equivalent subjectAltName rfc822Name values and NETCONF
        username values except that the domain-part of the name
        MUST be passed in lowercase.

        Example rfc822Name Field:  FooBar@Example.COM
        is mapped to NETCONF username: FooBar@example.com.";
    }
    enum dNSName {
      description
        "Maps a subjectAltName's dNSName to a NETCONF username after
        first converting it to all lowercase (RFC 5280 does not
        specify converting to lowercase so this involves an extra
        step). This mapping results in a 1:1 correspondence between
        subjectAltName dNSName values and the NETCONF username
        values.

        reference:  RFC 5280 - Internet X.509 Public Key
                    Infrastructure Certificate and Certificate
                    Revocation List (CRL) Profile.";
    }
    enum ipAddress {
      description
        "Maps a subjectAltName's ipAddress to a NETCONF username by
        transforming the binary encoded address as follows:

        1) for IPv4, the value is converted into a
           decimal-dotted quad address (e.g., '192.0.2.1').

        2) for IPv6 addresses, the value is converted into a
           32-character all lowercase hexadecimal string
           without any colon separators.

        This mapping results in a 1:1 correspondence between
        subjectAltName ipAddress values and the NETCONF username
        values.";
    }
  }
} // leaf-list from-certificate
```

```
description
  "Specifies the algorithm for deriving a NETCONF username from
  a certificate.  If a mapping succeeds, then it will return a
  NETCONF username.

  If the resulting mapped value is not compatible with the
  needed requirements of a NETCONF username, then subsequent
  cert-map list entries MUST be searched for additional
  matches to look for a mapping that succeeds.";

} // choice map-type
} // list cert-map
} // container cert-maps

//
// Objects related to deriving NETCONF usernames from TLS pre-shared
// keys.
//

container psk-maps {
  if-feature map-pre-shared-keys;

  description
    "During the TLS Handshake, the client indicates which key to use
    by including a PSK identity in the TLS ClientKeyExchange message.
    On the server side, this PSK identity is used to look up an entry
    in the psk-map list.  If such an entry is found, and the pre-shared
    keys match, then the client is authenticated.  The server uses the
    value from the user-name leaf in the psk-map list as the NETCONF
    username.  If the server cannot find an entry in the psk-map list,
    or if the pre-shared keys do not match, then the server terminates
    the connection.  For details on how the PSK identity MAY be encoded
    in UTF-8, see section 5.1. of RFC 4279.";

  reference
    "RFC 4279: Pre-Shared Key Ciphersuites for Transport Layer
    Security (TLS)";

  list psk-map {
    key psk-identity;

    leaf psk-identity {
      type string;
      description
        "The PSK identity encoded as a UTF-8 string.";
      reference
        "RFC 4279: Pre-Shared Key Ciphersuites for Transport Layer
```



```

        Security (TLS)";
    }

    leaf user-name {
        type nacm:user-name-type;
        mandatory true;
        description
            "The NETCONF username associated with this PSK identity.";
    }

    leaf valid-not-before {
        type yang:date-and-time;
        description
            "This PSK identity is not valid before the given data
            and time.";
    }

    leaf valid-not-after {
        type yang:date-and-time;
        description
            "This PSK identity is not valid before the given date
            and time.";
    }

    leaf key {
        type string {
            pattern '([0-9a-fA-F]){2}(:([0-9a-fA-F]){2})*';
        }
        nacm:default-deny-all;
        description
            "The key associated with the PSK identity";
    }
} // list psk-map
} // container psk-maps
} // container tls
} // container netconf-config
}

```

#### 4. Security Considerations

The security considerations described throughout [RFC5246] and [RFC6241] apply here as well.

This document in its current version 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, SSH transport can be used.

An attacker might be able to inject arbitrary NETCONF messages via some application that does not carefully check exchanged messages. When the :base:1.1 capability is not advertised by both peers, an attacker might be able to deliberately insert the delimiter sequence `]]>]]>` in a NETCONF message to create a DoS attack. If the :base:1.1 capability is not advertised by both peers, applications and NETCONF APIs MUST ensure that the delimiter sequence `]]>]]>` never appears in NETCONF messages; otherwise, those messages can be dropped, garbled, or misinterpreted. More specifically, if the delimiter sequence is found in a NETCONF message by the sender side, a robust implementation of this document SHOULD warn the user that illegal characters have been discovered. If the delimiter sequence is found in a NETCONF message by the receiver side (including any XML attribute values, XML comments, or processing instructions), a robust implementation of this document MUST silently discard the message without further processing and then stop the NETCONF session.

Finally, this document does not introduce any new security considerations compared to [RFC6242].

## 5. 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 name "netconf-tls". This port will be the default port for NETCONF over TLS, as defined in this document.

Registration Contact: Mohamad Badra, mbadra@gmail.com.  
Transport Protocol: TCP.  
Port Number: 6513  
Broadcast, Multicast or Anycast: No.  
Port Name: netconf-tls.  
Service Name: netconf.  
Reference: RFC 5539

## 6. Acknowledgements

A significant amount of the text in Section 3 was lifted from [RFC4642].

The author would like to acknowledge David Harrington, Miao Fuyou, Eric Rescorla, Simon Josefsson, Olivier Coupelon, Alfred Hoenes, and the NETCONF mailing list members for their comments on the document. The author also appreciates Bert Wijnen, Mehmet Ersue, and Dan Romascanu for their efforts on issues resolving discussion; and Charlie Kaufman, Pasi Eronen, and Tim Polk for the thorough review of previous versions of this document.

## 7. Contributor's Address

Ibrahim Hajjeh  
Ineovation  
France

EEmail: [ibrahim.hajjeh@ineovation.fr](mailto:ibrahim.hajjeh@ineovation.fr)

Martin Bjorklund  
Tail-f Systems

Email: [mbj@tail-f.com](mailto:mbj@tail-f.com)

## 8. References

### 8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.
- [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.
- [RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, June 2011.
- [RFC6353] Hardaker, W., "Transport Layer Security (TLS) Transport Model for the Simple Network Management Protocol (SNMP)", RFC 6353, July 2011.

## 8.2. Informative References

- [RFC4642] Murchison, K., Vinocur, J., and C. Newman, "Using Transport Layer Security (TLS) with Network News Transfer Protocol (NNTP)", RFC 4642, October 2006.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, July 2008.
- [RFC5539] Badra, M., "NETCONF over Transport Layer Security (TLS)", RFC 5539, May 2009.
- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

## Appendix A. Change Log (to be removed by RFC Editor before publication)

### A.1. From draft-ietf-netconf-rfc5539bis-00 to draft-ietf-netconf-rfc5539bis-01

- o Update Section 3.2 and address some issues raised during WGLC

### A.2. From draft-badra-netconf-rfc5539bis-02 to draft-ietf-netconf-rfc5539bis-00

- o Remove the reference to BEEP
- o Rename host-part to domain-part in the description of RFC822.

Authors' Addresses

Mohamad Badra  
LIMOS Laboratory

Email: [mbadra@gmail.com](mailto:mbadra@gmail.com)

Alan Luchuk  
SNMP Research

Email: [luchuk@snmp.com](mailto:luchuk@snmp.com)

Juergen Schoenwaelder  
Jacobs University Bremen

Email: [j.schoenwaelder@jacobs-university.de](mailto:j.schoenwaelder@jacobs-university.de)

