

netconf
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2015

B. Liu
G. Zheng
Huawei Technologies
October 27, 2014

A NETCONF Extension for Data Fragmentation
draft-liu-netconf-fragmentation-01

Abstract

This document introduces an extension to NETCONF (Network Configuration) protocol. The extension allows NETCONF to handle large size data as fragmented RPC messages. Specifically, this document defines a new <get-block> capability and relevant operations to handle the fragmentations.

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 30, 2015.

Copyright Notice

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

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

Table of Contents

1. Introduction	2
2. Requirements Language and Terminology	3
3. Current Large size Handling Methods and Problems	3
3.1. Stream-Oriented Handling	3
3.2. Requesting a Portion of Data	4
4. Candidate Solutions	4
4.1. Netconf Fragmentation Mechanism	4
4.1.1. Fragmentation Requirements	4
4.1.2. <get-block> extention	5
4.2. Subtree Iteration	6
4.3. Linked Replies	6
5. Security Considerations	7
6. IANA Considerations	7
7. Acknowledgements	7
8. References	7
8.1. Normative References	7
8.2. Informative References	7
Appendix A. Examples of the Candidate Solutions	8
A.1. <get-next> (RPC Fragmentation) Example	8
A.2. <get-list> (Subtree Iteration) Example	10
A.3. Linked-replies Example	11
Authors' Addresses	12

1. Introduction

NETCONF [RFC6241] is the next generation network management protocol for configuring devices. It is becoming more and more popular, and some NMS (Network Management System) only use NETCONF as its southbound interface. The message procedures of NETCONF are based on RPC (Remote Procedure Call) interactions. A NETCONF client/server sends a <rpc> message to the counterpart and then receives a replying <rpc-reply> message.

In some situations, the <rpc-reply> message might be very large. For example, when NMS is retrieving a large amount of routes in a core router or doing a full-synchronizing with a device, the <rpc-reply> data might exceed Mega-Byte amount. Then there comes the problem of how to handle the large size data. In Section 3, this document briefly introduces two typical ways of current handling on this issue; and analyzes the problems of them.

To fix the problems, in Section 4, this document proposes a method of extending the NETCONF protocol to allow handling large size data as fragmented <rpc-reply> messages. The fragmentation is done at the NETCONF level, so it allows the NETCONF client to terminate the large size data processing momentarily by protocol interactions; and also

allows the fragmented messages to be instantly parsed piece by piece. Specifically, the fragmentation is achieved through a newly defined <get-block> capability and relevant operations.

2. Requirements Language and Terminology

The key words "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 [RFC2119] when they appear in ALL CAPS. When these words are not in ALL CAPS (such as "should" or "Should"), they have their usual English meanings, and are not to be interpreted as [RFC2119] key words.

Terminology:

DOM: Document Object Model, which is a cross-platform and language-independent convention for representing and interacting with objects in HTML, XHTML and XML documents. Objects in the DOM tree may be addressed and manipulated by using methods on the objects. [DOM]

SAX: Simple API for XML, which is an event sequential access parser API developed by the XML-DEV mailing list for XML documents. SAX provides a mechanism for reading data from an XML document that is an alternative to that provided by the DOM. Where the DOM operates on the document as a whole, SAX parsers operate on each piece of the XML document sequentially. [SAX]

libxml: a software library for parsing XML documents. [LIBXML]

<get-block>: a capability and operation defined in this document to handle large size

3. Current Large size Handling Methods and Problems

3.1. Stream-Oriented Handling

Stream-Oriented handling mainly includes the following two aspects:

- o The server encapsulates the large size replying data in a <rpc-reply> message and streams it to the client through TCP protocol.
- o The client parses the received <rpc-reply> content in a stream-oriented way. More specifically, the client could utilize SAX [SAX-PARSING] to instantly parse the received content without waiting for the whole message been transported.

Problems:

- o Stream-Oriented method lacks the capability of discontinuing large size processing in the server. It would cause unnecessary resource/performance cost in the devices if the NMS has already got the intended portion or just canceled by the administrators.
- o Another problem is the implementation of SAX parsing is more complex than DOM parsing [DOM-PARSING] in the Netconf client. More computing burden will be taken in Netconf client to support SAX parsing.

3.2. Requesting a Portion of Data

The clients actively limit the search range of the data so that the servers only need to reply with a part of the large size data. Thus the clients could control the replies in a reasonable size. One example is that the clients get a list of the content, and provide a start offset and a max-count, to get a portion at a time.

Problems:

- o This method has an implication that the client needs to know the list/index of the intended large size data in advance before it starting the search request. It can't fit the scenarios of real-time on-demand data retrieving. And there is no standard to specify the list/index format in a uniform way. Thus it is only suitable for private implementation, thus multi-vendor interaction is not supported.
- o More important, it is just an indirect way to solve the problem. It could not fit the scenarios where the client just needs the whole large size data in the server.

4. Candidate Solutions

(Editor notes: this section discusses several possible solutions. The fragmentation mechanism is the original proposal of the draft. The other two were proposed during mailing list discussion by Andy Bierman and Juergen Schoenwaelder respectively. We include all of them for discussion and solution selection.)

4.1. Netconf Fragmentation Mechanism

4.1.1. Fragmentation Requirements

this document proposes an RPC fragmentation mechanism to handle the large size data. Two essential requirements of the fragmentation are:

- o It needs to allow the NETCONF client to terminate the large size data processing momentarily by protocol interactions. In the proposed mechanisms in this draft, when the NETCONF server replies the client an <rpc-reply> fragmentation, it will wait the response from the client that whether it needs to send the next fragmentation. So if the initiator has got the intended portion, it could terminate the large size process immediately.
- o It needs to allow the NETCONF client to instantly parse the fragmentations piece by piece through the more widely supported DOM parsing. So in this document, it specifies that each <rpc-reply> fragmentation MUST be in a complete XML form.

4.1.2. <get-block> extention

- o Function

The devices can only use <get-block> operation when the Get-block capability was announced.

The <get-block> fragmentation rules are:

- A. There should be a Max-Size for fragmentation. [Open Question]Should there be a clear specification of the size? E.g. 64K bytes.
- B. When the message reaches the Max-Size, it is sent to the client and the next message could be created in advance.
- C. Different records from one same table could be put into different <rpc-reply> messages
- D. All of the fields in one record MUST be put into one <rpc-reply> message.
- E. XML syntax MUST be complete in each fragmented message, so that each fragmentation could be parsed individually.
- F. If the record(s) of the child node(s)/table(s) and the parent node(s)/table(s) are replied in different fragmentations, the child node/table fragmentations MUST include the path and index information of all the ancestor node(s)/table(s) in a hierarchical mode.

- o Parameters

<discard/>: in <get-block> operation; if the <discard/> parameter is conveyed, it means the operation is terminated. Then it doesn't need to reply the remaining fragmentations.

o Successful Operation Reply

A <rpc-reply> message conveying a <data> element indicates the operation is successful.

If there exists a next fragment, then an set-id attribute MUST be included in the <rpc-reply> message. The attribute set-id is used to identify different fragment sets.

o Exception Handling

After the NETCONF server replies a fragment, if there is no corresponding Get-block request from the client in a reasonable period (the time valued to be specified in the future), then the server release the offset of the replying data and cannot use <get-block> operation anymore, and the remaining data needs to be replied.

Please refer to Appendix A.1 for an example.

4.2. Subtree Iteration

An "iterator" approach allows a list resource to be retrieved in chunks. An RPC function could be added to do the iteration operation.

Please refer to Appendix A.2 for an example.

There is a problem with rapidly changing lists (could get repeat entries on miss some entries).

4.3. Linked Replies

Another solution is to change or augment NETCONF at some point in time such that an <rpc> can lead to a sequence of <rpc-reply> with a suitable cancel mechanism. A simple approach is to add a linked-replies capability. If a server announces "linked-replies" capability and the client supports it as well, the client can add an additional parameter to an rpc to indicate the possible use of linked-replies.

Please refer to Appendix A.3 for an example.

This would address the concern of large data retrievals but would also allow long running asynchronous rpcs (the ping or traceroute example). This approach may lead to better support for asynchronous rpcs and rpcs that potentially return very large chunks of data than trying to solve this problem without enhancements of the rpc layer. Design details concerning data merging, error handling, how to send a cancel for a given link-id (e.g., by sending a new <rpc-cancel> message with a matching link-id) and whether it is necessary to negotiate linked rpc-reply sizes or whether it is good enough for the server to decide freely as it likes etc. need further study.

5. Security Considerations

TBD.

6. IANA Considerations

This draft does not request any IANA action.

7. Acknowledgements

Gang Yan and Shouchuan Yang made significant contribution to form the draft. Valuable comments were received from Andy Bierman, Martin Bjorklund, Juergen Schoenwaelder, Chong Feng and some other people in Netconf working group.

This document was produced using the xml2rfc tool [RFC2629].
(initiallly prepared using 2-Word-v2.0.template.dot.)

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.
- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.
- [RFC6241] Enns, R., Bjorklund, M., Schoenwaelder, J., and A. Bierman, "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011.

8.2. Informative References

- [DOM] "http://en.wikipedia.org/wiki/Document_Object_Model", .

[DOM-PARSING] "http://www.w3.org/TR/DOM-Parsing/", .

[LIBXML] "http://xmlsoft.org/", .

[SAX] "http://en.wikipedia.org/wiki/Simple_API_for_XML", .

[SAX-PARSING] "http://www.saxproject.org/apidoc/org/xml/sax/
Parser.html", .

Appendix A. Examples of the Candidate Solutions

A.1. <get-next> (RPC Fragmentation) Example

Example 1: Get the next fragment

```
<rpc message-id="101"
  xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
  <get-config>
    <source>
      <running/>
    </source>
    <filter type="subtree">
      <top xmlns="http://example.com/schema/1.2/config">
        <users/>
      </top>
    </filter>
  </get-config>
</rpc>

<rpc-reply message-id="101"
  xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0"
  xmlns:hw="http://www.huawei.com/NETCONF/capability/base/1.0"
hw:set-id="101">
  <data>
    <top xmlns="http://example.com/schema/1.2/config">
      <users>
        <user>
          <name>root</name>
          <type>superuser</type>
          <full-name>Charlie Root</full-name>
          <company-info>
            <dept>1</dept>
            <id>1</id>
          </company-info>
        </user>
        <!-- additional <user> elements appear here... -->
      </users>
    </top>
  </data>
</rpc-reply>
```



```

        </top>
      </data>
    </rpc-reply>

    <rpc message-id="102"
      xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
      <get-block xmlns="http://www.huawei.com/NETCONF/capability/base/1.0"
set-id="1">
        </get-block>
      </rpc>
      <rpc-reply message-id="102"
        xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0"
        xmlns:hw="http://www.huawei.com/NETCONF/capability/base/1.0"
hw:set-id="101">
        <data>
          <top xmlns="http://example.com/schema/1.2/config">
            <users>
              <user>
                <name>admin</name>
                <type>commonuser</type>
                <full-name>Jim Green</full-name>
                <company-info>
                  <dept>9</dept>
                  <id>90</id>
                </company-info>
              </user>
              <!-- additional <user> elements appear here... -->
            </users>
          </top>
        </data>
      </rpc-reply>

```

Example 2: Abandon the remaining fragments

```

    <rpc message-id="103"
      xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
      <get-block xmlns="http://www.huawei.com/NETCONF/capability/base/1.0" set-id
="1">
        <discard/>
      </get-block>
    </rpc>

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

```

Example 3: Following is an example of the rule f in Section 4.1.2. The child eTable is in a different message with the parents aTable->bTable->cTable->dTable. Then the path and index information

of all the ancestors MUST included in the search data.

```
<aTable>
  <aEntity>
    <aIndex1>
  </aEntity>
  <bTable>
    <bEntity>
  <bIndex1>
</bEntity>
</dTable>
<eTable>
  <eEntity>
    <eIndex1>
    <ef2>
    <ef3>
  </eEntity>
</dTable>
</bTable>
</aTable>
```

A.2. <get-list> (Subtree Iteration) Example

```
rpc get-list {
  input {
    leaf target {
      type schema-instance-identifier;
      description "Identifies subtree to retrieve.";
    }
    leaf start {
      type uint32;
      default 0;
      description "Number of entries to skip before starting retrieval";
    }
    leaf max-entries {
      type uint32 { range "1..max"; }
      default 25;
      description "Maximum number of list entries to retrieve";
    }
  }
  output {
    anyxml data {
      description "Contains the requested data";
    }
  }
}

<rpc>
```

```
<get-list>
  <target>/if:interfaces/if:interface</target>
</get-list>
</rpc>

<rpc-reply>
  <data>
    <interfaces>
      <interface> .... first entry </interface>
      ...
      <interface> .... 25th entry </interface>
    </interfaces>
  </data>
</rpc-reply>

<rpc>
  <get-list>
    <target>/if:interfaces/if:interface</target>
    <start>25</start>
  </get-list>
</rpc>

<rpc-reply>
  <data>
    <interfaces>
      <interface> .... 26th entry </interface>
      ...
      <interface> .... 50th entry </interface>
    </interfaces>
  </data>
</rpc-reply>
```

A.3. Linked-replies Example

Here is what a new client might do if it wants to use linked replies:

```
<rpc message-id="101" link-id="123" xmlns="...">
</rpc>
```

The server can either simply send an rpc-reply or it starts sending linked replies, e.g.:

```
<rpc-reply message-id="101" next-message-id="102" link-id="123" xmlns="...">
</rpc-reply>
```

```
<rpc-reply message-id="102" next-message-id="103" link-id="123" xmlns="...">
</rpc-reply>
```

```
<rpc-reply message-id="103" link-id="123" xmlns="...">
</rpc-reply>
```

Authors' Addresses

Bing Liu
Huawei Technologies
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
P.R. China

Email: leo.liubing@huawei.com

Guangying Zheng
Huawei Technologies
Huawei Nanjing R&D Center
101 Software Avenue, Yuhua District, Nanjing, Jiangsu, 210012
P.R. China

Email: zhengguangying@huawei.com