netconf
Internet-Draft
Intended status: Standards Track
Expires: September 23, 2016

B. Liu
G. Zheng
Huawei Technologies
M. Jethanandani
Cisco Systems
K. Watsen
Juniper Networks

March 22, 2016

Processing Multiple Replies for One Request in NETCONF
draft-liu-netconf-multiple-replies-02

Abstract

This document discusses several scenarios that multiple replies for a single request are needed, with the ability to terminate the replies at any time. Such scenarios are not well supported by current NETCONF (Network Configuration) protocol. An extention at the NETCONF messaging layer is needed to fulfill the requirement.

Table of Contents

1. Introduction

The message procedures of NETCONF [RFC6241] 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, it might need multiple <rpc-reply> messages for a <rpc> request. For example, the the <rpc-reply> message might be very large that it needs to be fragmented into multiple small ones; or some operations (e.g. ping) need persistent replies till such time that a terminaing condition is encountered or when the operation is cancelled.

This document discusses such kind of multiple replies scenarios, analyzes the issues of current NETCONF protocol in supporting these scenarios, and proposes several candidate solutions for discussion.

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:

CLI: Command Line interface

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.

RPC: Remote Procedure Call

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.

libxml: a software library for parsing XML documents.

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

3. Scenarios and Problems

This section discusses several scenarios where multiple replies might be needed, and anylizes the problems of current NETCONF protocol in supporting these scenarios.

3.1. Bulk <rpc-reply>

The <rpc-reply> message might be very large in following situations:

- retrieving a large amount of routes in a core router
- retrieving a large interface statistics list
- doing a full-synchronizing with a device

Current there are already some methods of processing bulk replies as the following. However, there are some issues as analyzed below.

1) Stream-Oriented Handling

Stream-Oriented handling mainly includes the following two aspects:

- The server encapsulates the large size replying data in a <rpc-reply> message and streams it to the client through transport protocol.

- The client parses the received <rpc-reply> content in a stream-oriented way. More specifically, the client could utilize SAX parsing to instantly parse the received content without waiting for the whole message been transported.

The problems are:

- 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 NETCONF client has already got the intended portion or just canceled by the administrators.

- Another problem is the implementation of SAX parsing is more complex than DOM parsing in the NETCONF client. More computing burden will be taken to support SAX parsing.

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.

The problems are:

- 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.

- 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.

3.2. Persistent <rpc-reply>

One of the operations that CLI offers today is the ability to issue an operation that might result in multiple responses being returned, till such time that a terminaing condition is encountered or when the operation is cancelled. An example of such an operation is when the ping or a traceroute command is issued. In the former case, the operation can continue sending responses back till it is cancelled, while in the latter case there is usually a terminating condition that stops the responses.

NETCONF protocol as defined today sends a single RPC request and expects a single reply to that request. The "persistent" operation defined above expects multiple responses for a single request, till such time a terminating condition is encountered.

3.3. Long time <rpc-reply>

Some operations might take a long time to perform (e.g. network link performance validation), so there could be multiple responses for the request. For example, initial responses returns handle which the client uses to monitor progress till the final result. The client should be able to cancel the request at any time.

3.4. Datastore push updates

In [I-D.clemm-netconf-yang-push] and [I-D.ietf-i2rs-pub-sub-requirements], it describes a scenario where client applications need to request updates from a YANG datastore, without requiring additional client requests. [I-D.clemm-netconf-yang-push] proposes to extend notification messaging to fulfill the requirement that a comprehensive subsription/publication model could be well supported.

The datastore sub/pub might need specific data modeling and operation extention. However, at the NETCONF message layer, this draft considers multiple replies could be an alternative solution for sub/pub comparing with notification messaging extention.

[Open Question] Need more in-depth analysis and comparison of the two alternatives.

4. Requriments for NETCONF

Given above mentioned multiple-replies scenarios and problems, the requriments for NETCONF protocol could be summarized as a mechanism with the following abilities:

- be able to generate and handle multiple <rpc-reply> messages for a given <rpc> message
- be able to terminate the <rpc-reply> at any time
- be able to cancel the request in pipeline scenarios

5. Candidate Solutions

(Editor notes: this section discusses possible solutions. The fragmentation mechanism is from the draft [I-D.liu-netconf-fragmentation]. The other one was proposed during mailing list discussion by Juergen Schoenwaelder. We include both of them for discussion and solution selection.)

5.1. Get-block Mechanism

This section proposes a method of extending the NETCONF protocol to allow handling the replying data as multiple <rpc-reply> blocks. It allows the NETCONF client to terminate the block data processing momentarily by protocol interactions; and also allows the blocked messages to be instantly parsed piece by piece. Specifically, the mechanism is achieved by a newly defined <get-block> capability and relevant operations.

5.1.1. Design Requirements

Two essential requirements of the Get-block mechanism are:

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

(Editor notes: this solution was originally designed for large size fragmentation processing. However, the rationale of this solution has the potential to fulfill the other scenarios.)

5.1.2. <get-block> extention

o Function

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

The <get-block> rules are:

A. There should be a Max-Size for each block. [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 block message, so that each block 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 blocks, the child node/table block 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 blocks.

o Successful Operation Reply

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

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

o Exception Handling

After the NETCONF server replies a block, 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.

5.2. 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" capablility 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.2 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.

6. Security Considerations

TBD.

7. IANA Considerations

This draft does not request any IANA action.

8. Acknowledgements

Gang Yan and Shouchuan Yang made significant contribution to form the draft. Valuable comments were received from Andy Bierman, Juergen Schoenwaelder, Balazs Lengyel, Martin Bjorklund, 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.)

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, DOI 10.17487/RFC2629, June 1999, <http://www.rfc-editor.org/info/rfc2629>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

9.2. Informative References

[I-D.clemm-netconf-yang-push]
Clemm, A., Prieto, A., and E. Voit, "Subscribing to YANG datastore push updates", draft-clemm-netconf-yang-push-02 (work in progress), October 2015.

[I-D.ietf-i2rs-pub-sub-requirements]
Voit, E., Clemm, A., and A. Prieto, "Requirements for Subscription to YANG Datastores", draft-ietf-i2rs-pub-sub-requirements-05 (work in progress), February 2016.

[I-D.liu-netconf-fragmentation]
Liu, B. and G. Zheng, "A NETCONF Extension for Data Fragmentation", draft-liu-netconf-fragmentation-01 (work in progress), October 2014.

Appendix A. Examples of the Candidate Solutions

A.1. <get-block> Example

Example 1: Get the next block

```
     <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://example.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>

<rpc message-id="102"
          xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
       <get-block xmlns="http://example.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://example.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 blocks

```
    <rpc message-id="103"
         xmlns="urn:ietf:params:xml:ns:NETCONF:base:1.0">
      <get-block xmlns=http://example.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>
<eTable>
  <eEntity>
    <eIndex1>
    <ef2>
    <ef3>
  </eEntity>
</dTable>
  </bTable>
</aTable>
```

A.2. 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="101" link-id="123" xmlns="...">
</rpc-reply>

<rpc-reply message-id="101" next-message-id="101" link-id="124" xmlns="...">
</rpc-reply>

<rpc-reply message-id="101" link-id="125" 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

Mahesh Jethanandani
Cisco Systems
USA

Email: mjethanandani@gmail.com

Kent Watsen
Juniper Networks

Email: kwatsen@juniper.net