Network File System Version 4 C. Lever, Ed.
Internet-Draft Oracle
Intended status: Standards Track D. Noveck
Expires: December 5, 2016 HPE
June 3, 2016

RPC-over-RDMA Version Two Protocol
draft-cel-nfsv4-rpcrdma-version-two-01

Abstract

This document specifies an improved protocol for conveying Remote Procedure Call (RPC) messages on physical transports capable of Remote Direct Memory Access (RDMA), based on RPC-over-RDMA Version One.

Table of Contents

1. Introduction

Remote Direct Memory Access (RDMA) [RFC5040] [RFC5041] [IB] is a technique for moving data efficiently between end nodes. By directing data into destination buffers as it is sent on a network, and placing it via direct memory access by hardware, the complementary benefits of faster transfers and reduced host overhead are obtained.

A protocol already exists that enables ONC RPC [RFC5531] messages to be conveyed on RDMA transports. That protocol is RPC-over-RDMA Version One, specified in [I-D.ietf-nfsv4-rfc5666bis]. RPC-over-RDMA Version One is deployed and in use, though there are some shortcomings to this protocol, such as:

o The use of small Receive buffers force the use of RDMA Read and Write transfers for small payloads, and limit the size of backchannel messages

o Lack of support for potential optimizations, such as remote invalidation, that require changes to on-the-wire behavior

To address these issues in a way that is compatible with existing RPC-over-RDMA Version One deployments, a new version of RPC-over-RDMA is presented in this document. RPC-over-RDMA Version Two contains only incremental changes over RPC-over-RDMA Version One to facilitate adoption of Version Two by existing Version One implementations.

The major new feature in RPC-over-RDMA Version Two is extensibility of the RPC-over-RDMA header. Extensibility enables narrow changes to RPC-over-RDMA Version Two so that new optional capabilities can be introduced without a protocol version change and while maintaining interoperability with existing implementations. New capabilities can be proposed and developed independently of each other, and implementaters can choose among them. It should be straightforward to create and document experimental features and then bring them through the standards process.

In addition to extensibility, the default inline threshold value is larger in RPC-over-RDMA Version Two. This change is driven by the increase in average size of RPC messages containing common NFS operations. With NFSv4.1 [RFC5661] and later, compound operations convey more data per RPC message. The default 1KB inline threshold in RPC-over-RDMA Version One prevents attaining the best possible performance.

1.1. Requirements Language

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

2. Inline Threshold

2.1. Terminology

The term "inline threshold" is defined in Section 4 of [I-D.ietf-nfsv4-rfc5666bis]. An "inline threshold" value is the largest message size (in octets) that can be conveyed in one direction between peer implementations using RDMA Send and Receive. Each connection has two inline threshold values: one for messages flowing from requester-to-responder (referred to as the "call inline threshold"), and one for messages flowing from responder-to-requester (referred to as the "reply inline threshold"). Inline threshold values are not advertised to peers via the base RPC-over-RDMA Version Two protocol.

A connection's inline threshold determines when full RDMA Read or Write operations are required because the RPC message to be send is larger than the peer's Receive buffer. If an RPC message does not contain DDP-eligible data items, a requester prepares a Long Call or Reply to convey the whole RPC message using an RDMA Read or Write.

2.2. Motivation

RDMA Read and Write operations require that each data payload resides in a region of memory that is registered with the RNIC. When an RPC is complete, that region is unregistered, fencing it from the responder.

Both registration and unregistration have a latency cost which is insignificant compared to data handling costs. When a data payload is small, however, the cost of registering and unregistering the memory where it resides becomes a relatively significant part of total RPC latency. Therefore the most efficient operation of RPC-over-RDMA occurs when RDMA Read and Write operations are used for large payloads, and avoided for small payloads.

When RPC-over-RDMA Version One was conceived, the typical size of RPC messages that did not involve a significant data payload was under 500 bytes. A 1024-byte inline threshold adequately minimized the frequency of inefficient Long Calls and Replies.

Starting with NFSv4.1 [RFC5661], NFS COMPOUND RPC messages are larger and more complex than before. With a 1024-byte inline threshold, RDMA Read or Write operations are needed for frequent operations that do not bear a data payload, such as GETATTR and LOOKUP, reducing the efficiency of the transport. To reduce the need to use Long Calls and Replies, RPC-over-RDMA Version Two quadruples the default inline threshold size. This also increases the maximum size of backward direction RPC messages.

2.3. Default Values

RPC-over-RDMA Version Two receiver implementations MUST support an inline threshold of 4096 bytes, but MAY support larger inline threshold values. A mechanism for discovering a peer's preferred inline threshold value may be used to optimize RDMA Send operations further. In the absense of such a mechanism, senders MUST assume a receiver's inline threshold is 4096 bytes.

The new default inline threshold size is no larger than the size of a hardware page on typical platforms. This conserves the resources needed to Send and Receive base level RPC-over-RDMA Version Two messages, enabling RPC-over-RDMA Version Two to be used on a broad base of hardware.

3. Protocol Extensibility

The core RPC-over-RDMA Version Two header format is specified in Section 4 as a complete and stand-alone piece of XDR. Any change to this XDR description requires a protocol version number change.

3.1. Optional Features

RPC-over-RDMA Version Two introduces the ability to extend the core protocol via optional features. Extensibility enables minor protocol issues to be addressed and incremental enhancements to be made without the need to change the protocol version. The key capability is that both sides can detect whether a feature is supported by their peer or not. With this ability, OPTIONAL features can be introduced over time to an otherwise stable protocol.

The rdma_opttype field carries a 32-bit unsigned integer. The value in this field denotes an optional operation that MAY be supported by the receiver. The values of this field and their meaning are defined in other Standards Track documents.

The rdma_optinfo field carries opaque data. The content of this field is data meaningful to the optional operation denoted by the value in rdma_opttype. The content of this field is not defined in the base RPC-over-RDMA Version Two protocol, but is defined in other Standards Track documents

When an implementation does not recognize or support the value contained in the rdma_opttype field, it MUST send an RPC-over-RDMA message with the rdma_xid field set to the same value as the erroneous message, the rdma_proc field set to RDMA2_ERROR, and the rdma_err field set to RDMA2_ERR_INVAL_OPTION.

3.2. Message Direction

Backward direction operation depends on the ability of the receiver to distinguish between incoming forward and backward direction calls and replies. This needs to be done because both the XID field and the flow control value (RPC-over-RDMA credits) in the RPC-over-RDMA header are interpreted in the context of each message's direction.

A receiver typically distinguishes message direction by examining the mtype field in the RPC header of each incoming payload message. However, RDMA2_OPTIONAL type messages may not carry an RPC message payload.

To enable RDMA2_OPTIONAL type messages that do not carry an RPC message payload to be interpreted unambiguously, the rdma2_optional structure contains a field that identifies the message direction. A similar field has been added to the rpcrdma2_chunks and rpcrdma2_error structures to simplify parsing the RPC-over-RDMA header at the receiver.

3.3. Documentation Requirements

RPC-over-RDMA Version Two may be extended by defining a new rdma_opttype value, and then by providing an XDR description of the rdma_optinfo content that corresponds with the new rdma_opttype value. As a result, a new header type is effectively created.

A Standards Track document introduces each set of such protocol elements. Together these elements are considered an OPTIONAL feature. Each implementation is either aware of all the protocol elements introduced by that feature, or is aware of none of them.

Documents describing extensions to RPC-over-RDMA Version Two should contain:

- An explanation of the purpose and use of each new protocol element added

- An XDR description of the protocol elements, and a script to extract it

- A mechanism for reporting errors when the error is outside the available choices already available in the base protocol or in other extensions

- An indication of whether a Payload stream must be present, and a description of its contents

- A description of interactions with existing extensions

The last bullet includes requirements that another OPTIONAL feature needs to be present for new protocol elements to work, or that a particular level of support be provided for some particular facility for the new extension to work.

Implementers combine the XDR descriptions of the new features they intend to use with the XDR description of the base protocol in this document. This may be necessary to create a valid XDR input file because extensions are free to use XDR types defined in the base protocol, and later extensions may use types defined by earlier extensions.

The XDR description for the RPC-over-RDMA Version Two protocol combined with that for any selected extensions should provide an adequate human-readable description of the extended protocol.

4. XDR Protocol Definition

This section contains a description of the core features of the RPC-over-RDMA Version Two protocol, expressed in the XDR language [RFC4506].

This description is provided in a way that makes it simple to extract into ready-to-compile form. The reader can apply the following shell script to this document to produce a machine-readable XDR description of the RPC-over-RDMA Version One protocol without any OPTIONAL extensions.

```
<CODE BEGINS>

#!/bin/sh
grep '^ *///' | sed 's?^ /// ??' | sed 's?^ *///$??'

<CODE ENDS>
```

That is, if the above script is stored in a file called "extract.sh" and this document is in a file called "spec.txt" then the reader can do the following to extract an XDR description file:

```
<CODE BEGINS>

sh extract.sh < spec.txt > rpcrdma_corev2.x

<CODE ENDS>
```

Optional extensions to RPC-over-RDMA Version Two, published as Standards Track documents, will have similar means of providing XDR that describes those extensions. Once XDR for all desired extensions is also extracted, it can be appended to the XDR description file extracted from this document to produce a consolidated XDR description file reflecting all extensions selected for an RPC-over-RDMA implementation.

4.1. Code Component License

Code components extracted from this document must include the following license text. When the extracted XDR code is combined with other complementary XDR code which itself has an identical license, only a single copy of the license text need be preserved.

<CODE BEGINS>

```
/// /*
///  * Copyright (c) 2010, 2016 IETF Trust and the persons
///  * identified as authors of the code.  All rights reserved.
///  *
///  * The authors of the code are:
///  * B. Callaghan, T. Talpey, C. Lever, and D. Noveck.
///  *
///  * Redistribution and use in source and binary forms, with
///  * or without modification, are permitted provided that the
///  * following conditions are met:
///  *
///  * - Redistributions of source code must retain the above
///  *   copyright notice, this list of conditions and the
///  *   following disclaimer.
///  *
///  * - Redistributions in binary form must reproduce the above
///  *   copyright notice, this list of conditions and the
///  *   following disclaimer in the documentation and/or other
///  *   materials provided with the distribution.
///  *
///  * - Neither the name of Internet Society, IETF or IETF
///  *   Trust, nor the names of specific contributors, may be
///  *   used to endorse or promote products derived from this
///  *   software without specific prior written permission.
///  *
///  *   THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS
///  *   AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED
///  *   WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
///  *   IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
///  *   FOR A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO
///  *   EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
///  *   LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
///  *   EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
///  *   NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
///  *   SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
///  *   INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
///  *   LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
///  *   OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING
///  *   IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
///  *   ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
///  */
```

<CODE ENDS>

4.2. RPC-Over-RDMA Version Two XDR

The XDR defined in this section is used to encode the Transport Header Stream in each RPC-over-RDMA Version Two message. The terms "Transport Header Stream" and "RPC Payload Stream" are defined in Section 4 of [I-D.ietf-nfsv4-rfc5666bis].

```
<CODE BEGINS>

/// /* From RFC 5531, Section 9 */
/// enum msg_type {
///         CALL = 0,
///         REPLY = 1
/// };
///
/// struct rpcrdma2_segment {
///         uint32 rdma_handle;
///         uint32 rdma_length;
///         uint64 rdma_offset;
/// };
///
/// struct rpcrdma2_read_segment {
///         uint32                  rdma_position;
///         struct rpcrdma2_segment rdma_target;
/// };
///
/// struct rpcrdma2_read_list {
///         struct rpcrdma2_read_segment rdma_entry;
///         struct rpcrdma2_read_list    *rdma_next;
/// };
///
/// struct rpcrdma2_write_chunk {
///         struct rpcrdma2_segment rdma_target<>;
/// };
///
/// struct rpcrdma2_write_list {
///         struct rpcrdma2_write_chunk rdma_entry;
///         struct rpcrdma2_write_list  *rdma_next;
/// };
///
/// struct rpcrdma2_chunk_lists {
///         enum msg_type               rdma_direction;
///         struct rpcrdma2_read_list   *rdma_reads;
///         struct rpcrdma2_write_list  *rdma_writes;
///         struct rpcrdma2_write_chunk *rdma_reply;
/// };
///
```

```
   /// enum rpcrdma2_errcode {
   ///         RDMA2_ERR_VERS = 1,
   ///         RDMA2_ERR_BAD_HEADER = 2,
   ///         RDMA2_ERR_INVAL_OPTION = 3
   /// };
   ///
   /// struct rpcrdma2_err_vers {
   ///         uint32 rdma_vers_low;
   ///         uint32 rdma_vers_high;
   /// };
   ///
   /// union rpcrdma2_error switch (rpcrdma2_errcode rdma_err) {
   ///         case RDMA2_ERR_VERS:
   ///           rpcrdma2_err_vers rdma_vrange;
   ///         case RDMA2_ERR_BAD_HEADER:
   ///           void;
   ///         case RDMA2_ERR_INVAL_OPTION:
   ///           void;
   /// };
   ///
   /// struct rpcrdma2_optional {
   ///         enum msg_type rdma_optdir;
   ///         uint32 rdma_opttype;
   ///         opaque rdma_optinfo<>;
   /// };
   ///
   /// enum rpcrdma2_proc {
   ///         RDMA2_MSG = 0,
   ///         RDMA2_NOMSG = 1,
   ///         RDMA2_ERROR = 4,
   ///         RDMA2_OPTIONAL = 5
   /// };
   ///
   /// union rpcrdma2_body switch (rpcrdma2_proc rdma_proc) {
   ///         case RDMA2_MSG:
   ///           rpcrdma2_chunks rdma_chunks;
   ///         case RDMA2_NOMSG:
   ///           rpcrdma2_chunks rdma_chunks;
   ///         case RDMA2_ERROR:
   ///           rpcrdma2_error rdma_error;
   ///         case RDMA2_OPTIONAL:
   ///           rpcrdma2_optional rdma_optional;
   /// };
   ///
   /// struct rpcrdma2_xprt_hdr {
   ///         uint32        rdma_xid;
   ///         uint32        rdma_vers;
   ///         uint32        rdma_credit;
```

```
   ///         rpcrdma2_body rdma_body;
   /// };

   <CODE ENDS>
```

4.2.1. Presence Of Payload

- o When the rdma_proc field has the value RDMA2_MSG, an RPC Payload Stream MUST follow the Transport Header Stream in the Send buffer.

- o When the rdma_proc field has the value RDMA2_ERROR, an RPC Payload Stream MUST NOT follow the Transport Header Stream.

- o When the rdma_proc field has the value RDMA2_OPTIONAL, all, part of, or no RPC Payload Stream MAY follow the Transport header Stream in the Send buffer.

4.2.2. Message Direction

Implementations of RPC-over-RDMA Version Two are REQUIRED to support backwards direction operation as described in [I-D.ietf-nfsv4-rpcrdma-bidirection].

- o When the rdma_proc field has the value RDMA2_MSG or RDMA2_NOMSG, the value of the rdma_direction field MUST be the same as the value of the associated RPC message's msg_type field.

- o When the rdma_proc field has the value RDMA2_ERROR, the direction of the message is always Responder-to-Requester (REPLY).

- o When the rdma_proc field has the value RDMA2_OPTIONAL and a whole or partial RPC message payload is present, the value of the rdma_optdir field MUST be the same as the value of the associated RPC message's msg_type field.

- o When the rdma_proc field has the value RDMA2_OPTIONAL and no RPC message payload is present, a Requester MUST set the value of the rdma_optdir field to CALL, and a Responder MUST set the value of the rdma_optdir field to REPLY. The Requester chooses a value for the rdma_xid field from the XID space that matches the message's direction. Requesters and Responders set the rdma_credit field in a similar fashion: a value is set that is appropriate for the direction of the message.

4.2.3. Transport Errors

Error handling works the same way in RPC-over-RDMA Version Two as it does in RPC-over-RDMA Version One, with one change described in Section 3.1. Version One error handling is described in Section 5 of [I-D.ietf-nfsv4-rfc5666bis].

5. Protocol Version Negotiation

When an RPC-over-RDMA Version Two requester establishes a connection to a responder, the first order of business is to determine the responder's highest supported protocol version.

As with RPC-over-RDMA Version One, a requester MUST assume the ability to exchange only a single RPC-over-RDMA message at a time until it receives a non-error RPC-over-RDMA message from the responder that reports the responder's actual credit limit.

First, the requester sends a single valid RPC-over-RDMA message with the value two (2) in the rdma_vers field. Because the responder might support only RPC-over-RDMA Version One, this initial message can be no larger than the Version One default inline threshold of 1024 bytes.

5.1. Responder Does Support RPC-over-RDMA Version Two

If the responder does support RPC-over-RDMA Version Two, it sends an RPC-over-RDMA message back to the requester with the same XID containing a valid non-error response. Subsequently, both peers use the default inline threshold value for RPC-over-RDMA Version Two connections (4096 bytes).

5.2. Responder Does Not Support RPC-over-RDMA Version Two

If the responder does not support RPC-over-RDMA Version Two, [I-D.ietf-nfsv4-rfc5666bis] REQUIRES that it send an RPC-over-RDMA message to the requester with the same XID, with RDMA2_ERROR in the rdma_proc field, and with the error code RDMA2_ERR_VERS. This message also reports a range of protocol versions that the responder supports. To continue operation, the requester selects a protocol version in the range of responder-supported versions for subsequent messages on this connection.

If the connection is lost immediately after the RDMA2_ERROR reply is received, a requester can avoid a possible version negotiation loop when re-establishing another connection by assuming that particular responder does not support RPC-over-RDMA Version Two. A requester can assume the same situation (no responder support for RPC-over-RDMA Version Two) if the initial negotiation message is lost or dropped.

Once the negotiation exchange is complete, both peers use the default inline threshold value for the protocol version that will be used for the remainder of the connection lifetime. To permit inline threshold values to change during negotiation of protocol version, RPC-over-RDMA Version Two implementations MUST allow inline threshold values to change without triggering a connection loss.

5.3. Requester Does Not Support RPC-over-RDMA Version Two

[I-D.ietf-nfsv4-rfc5666bis] REQUIRES that a responder MUST send Replies with the same RPC-over-RDMA protocol version that the requester uses to send its Calls.

6. Security Considerations

The security considerations for RPC-over-RDMA Version Two are the same as those for RPC-over-RDMA Version One.

7. IANA Considerations

There are no IANA considerations at this time.

8. Acknowledgments

The authors gratefully acknowledge the work of Brent Callaghan and Tom Talpey on the original RPC-over-RDMA Version One specification [RFC5666]. The authors also wish to thank Bill Baker, Greg Marsden, and Matt Benjamin for their support of this work.

The extract.sh shell script and formatting conventions were first described by the authors of the NFSv4.1 XDR specification [RFC5662].

Special thanks go to nfsv4 Working Group Chair Spencer Shepler and nfsv4 Working Group Secretary Thomas Haynes for their support.

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>.

[RFC4506] Eisler, M., Ed., "XDR: External Data Representation Standard", STD 67, RFC 4506, DOI 10.17487/RFC4506, May 2006, <http://www.rfc-editor.org/info/rfc4506>.

[RFC5531] Thurlow, R., "RPC: Remote Procedure Call Protocol Specification Version 2", RFC 5531, DOI 10.17487/RFC5531, May 2009, <http://www.rfc-editor.org/info/rfc5531>.

9.2. Informative References

[I-D.ietf-nfsv4-rfc5666bis] Lever, C., Simpson, W., and T. Talpey, "Remote Direct Memory Access Transport for Remote Procedure Call, Version One", draft-ietf-nfsv4-rfc5666bis-07 (work in progress), May 2016.

[I-D.ietf-nfsv4-rpcrdma-bidirection] Lever, C., "Bi-directional Remote Procedure Call On RPC-over-RDMA Transports", draft-ietf-nfsv4-rpcrdma-bidirection-04 (work in progress), May 2016.

[IB] InfiniBand Trade Association, "InfiniBand Architecture Specifications", <http://www.infinibandta.org>.

[RFC5040] Recio, R., Metzler, B., Culley, P., Hilland, J., and D. Garcia, "A Remote Direct Memory Access Protocol Specification", RFC 5040, DOI 10.17487/RFC5040, October 2007, <http://www.rfc-editor.org/info/rfc5040>.

[RFC5041] Shah, H., Pinkerton, J., Recio, R., and P. Culley, "Direct Data Placement over Reliable Transports", RFC 5041, DOI 10.17487/RFC5041, October 2007, <http://www.rfc-editor.org/info/rfc5041>.

[RFC5661] Shepler, S., Ed., Eisler, M., Ed., and D. Noveck, Ed., "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 5661, DOI 10.17487/RFC5661, January 2010, <http://www.rfc-editor.org/info/rfc5661>.

[RFC5662] Shepler, S., Ed., Eisler, M., Ed., and D. Noveck, Ed., "Network File System (NFS) Version 4 Minor Version 1 External Data Representation Standard (XDR) Description", RFC 5662, DOI 10.17487/RFC5662, January 2010, <http://www.rfc-editor.org/info/rfc5662>.

[RFC5666] Talpey, T. and B. Callaghan, "Remote Direct Memory Access Transport for Remote Procedure Call", RFC 5666, DOI 10.17487/RFC5666, January 2010, <http://www.rfc-editor.org/info/rfc5666>.

Authors' Addresses

Charles Lever (editor)
Oracle Corporation
1015 Granger Avenue
Ann Arbor, MI 48104
USA

Phone: +1 734 274 2396
Email: chuck.lever@oracle.com

David Noveck
Hewlett Packard Enterprise
165 Dascomb Road
Andover, MA 01810
USA

Phone: +1 978 474 2011
Email: davenoveck@gmail.com