

Network File System Version 4
Internet-Draft
Intended status: Standards Track
Expires: November 7, 2019

C. Lever, Ed.
Oracle
D. Noveck
NetApp
May 6, 2019

RPC-over-RDMA Version 2 Protocol
draft-cel-nfsv4-rpcrdma-version-two-09

Abstract

This document specifies a new version of the transport protocol that conveys Remote Procedure Call (RPC) messages on physical transports capable of Remote Direct Memory Access (RDMA). The new version of this protocol is extensible.

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 <https://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 November 7, 2019.

Copyright Notice

Copyright (c) 2019 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 (<https://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.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1.	Introduction	3
2.	Requirements Language	4
3.	RPC-over-RDMA Version 2 Headers and Chunks	5
3.1.	rpcrdma_common: Common Transport Header Prefix	5
3.2.	rpcrdma2_hdr_prefix: Version 2 Transport Header Prefix	6
3.3.	rpcrdma2_chunk_lists: Describe External Data Payload	7
4.	Transport Properties	8
4.1.	Transport Properties Model	8
4.2.	Current Transport Properties	10
4.2.1.	Receive Buffer Size	11
4.2.2.	Reverse Request Support	12
5.	RPC-over-RDMA Version 2 Transport Messages	13
5.1.	Overall Transport Message Structure	13
5.2.	Transport Header Types	13
5.3.	Header Types Defined in RPC-over-RDMA version 2	14
5.3.1.	RDMA2_MSG: Convey RPC Message Inline	15
5.3.2.	RDMA2_NOMSG: Convey External RPC Message	15
5.3.3.	RDMA2_ERROR: Report Transport Error	15
5.3.4.	RDMA2_CONNPROP: Advertise Transport Properties	18
6.	XDR Protocol Definition	19
6.1.	Code Component License	20
6.2.	Extraction and Use of XDR Definitions	22
6.3.	XDR Definition for RPC-over-RDMA Version 2 Core Structures	24
6.4.	XDR Definition for RPC-over-RDMA Version 2 Base Header Types	26
6.5.	Use of the XDR Description Files	27
7.	Protocol Version Negotiation	29
7.1.	Server Does Support RPC-over-RDMA Version 2	29

7.2.	Server Does Not Support RPC-over-RDMA Version 2	29
7.3.	Client Does Not Support RPC-over-RDMA Version 2	30
8.	Differences from the RPC-over-RDMA Version 1 Protocol	30
8.1.	Transport Properties	30
8.2.	Credit Management Changes	30

8.3.	Inline Threshold Changes	32
8.4.	Support for Remote Invalidation	33
8.4.1.	Reverse Direction Remote Invalidation	33
8.5.	Error Reporting Changes	34
9.	Extending the Version 2 Protocol	34
9.1.	Adding New Header Types to RPC-over-RDMA Version 2	35
9.2.	Adding New Transport properties to the Protocol	36
9.3.	Adding New Error Codes to the Protocol	37
9.4.	Adding New Header Flags to the Protocol	38
10.	Relationship to other RPC-over-RDMA Versions	38
10.1.	Relationship to RPC-over-RDMA Version 1	38
10.2.	Extensibility Beyond RPC-over-RDMA Version 2	40
11.	Security Considerations	40
11.1.	Security Considerations (Transport Properties)	40
12.	IANA Considerations	41
13.	References	41
13.1.	Normative References	41
13.2.	Informative References	41
	Acknowledgments	42
	Authors' Addresses	42

[1.](#) Introduction

Remote Direct Memory Access (RDMA) [[RFC5040](#)] [[RFC5041](#)] [[IBARCH](#)] 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 using direct memory access implemented by hardware, the complementary benefits of faster transfers and reduced host overhead are obtained.

RPC-over-RDMA version 1 enables ONC RPC [[RFC5531](#)] messages to be conveyed on RDMA transports. That protocol is specified in [[RFC8166](#)]. RPC-over-RDMA version 1 is deployed and in use, although there are known shortcomings to this protocol:

- o The protocol's default size of Receive buffers forces the use of

RDMA Read and Write transfers for small payloads, and limits the size of reverse direction messages.

- o It is difficult to make optimizations or protocol fixes that require changes to on-the-wire behavior.

To address these issues in a way that is compatible with existing RPC-over-RDMA version 1 deployments, a new version of the RPC-over-RDMA transport protocol is presented in this document.

This new version of RPC-over-RDMA is extensible, enabling OPTIONAL extensions to be added without impacting existing implementations.

To enable protocol extension, the XDR definition for RPC-over-RDMA version 2 is organized differently than the definition version 1. These changes, which are discussed in [Section 10.1](#), do not affect the on-the-wire format.

In addition, RPC-over-RDMA version 2 contains a set of incremental changes that relieve certain performance constraints and enable recovery from certain abnormal corner cases. These changes include:

- o The exchange of transport properties as described in [Section 8.1](#).
- o A more flexible credit account mechanism, detailed in Section TBD.
- o Larger default inline thresholds as described in [Section 8.3](#).
- o Support for remote invalidation as explained in [Section 8.4](#).
- o Support for reverse direction operation, as described in [\[RFC8167\]](#), is now REQUIRED. Details are in [Section 3.2](#).
- o An expansion of error reporting capabilities, described in [Section 5.3.3](#). A summary of the reasons for this expansion appears in [Section 8.5](#). This expansion supports the addition of new error codes as described in [Section 9.3](#).

Because of the way in which RPC-over-RDMA version 2 builds upon the facilities present in RPC-over-RDMA version 1, a knowledge of the basic structure of RPC-over-RDMA version 1, as described in [\[RFC8166\]](#), is assumed in this document.

As in that document, the terms "RPC Payload Stream" and "Transport Header Stream" (defined in [Section 3.2](#) of that document) are used to distinguish between an RPC message as defined by [\[RFC5531\]](#) and the header whose job it is to describe the RPC message and its associated memory resources. In that regard, the reader is assumed to understand how RDMA is used to transfer chunks between client and server, the use of Position-Zero Read chunks and Reply chunks to convey Long RPC messages, and the role of DDP-eligibility in constraining how data payloads are to be conveyed.

[2.](#) Requirements Language

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 [BCP 14](#) [\[RFC2119\]](#) [\[RFC8174\]](#) when, and only when, they appear in all capitals, as shown here.

Lever & Noveck

Expires November 7, 2019

[Page 4]

Internet-Draft

RDMA Transport for RPC V2

May 2019

[3.](#) RPC-over-RDMA Version 2 Headers and Chunks

Most RPC-over-RDMA version 2 data structures are derived from corresponding structures in RPC-over-RDMA version 1. As is typical for new versions of an existing protocol, the XDR data structures have new names and there are a few small changes in content. In some cases, there have been structural re-organizations to enabled protocol extensibility.

[3.1.](#) rpcrdma_common: Common Transport Header Prefix

The rpcrdma_common prefix describes the first part of each RDMA-over-RPC transport header for version 2 and subsequent versions.

<CODE BEGINS>

```
struct rpcrdma_common {
    uint32      rdma_xid;
    uint32      rdma_vers;
    uint32      rdma_credit;
    uint32      rdma_htype;
};
```

<CODE ENDS>

RPC-over-RDMA version 2's use of these first four words matches that of version 1 as required by [\[RFC8166\]](#). However, there are important structural differences in the way that these words are described by the respective XDR descriptions:

- o The header type is represented as a uint32 rather than as an enum that would need to be modified to reflect additions to the set of header types made by later extensions.
- o The header type field is part of an XDR structure devoted to representing the transport header prefix, rather than being part of a discriminated union, that includes the body of each transport header type.
- o There is now a prefix structure (see [Section 3.2](#)) of which the `rpcrdma_common` structure is the initial segment. This is a newly defined XDR object within the protocol description, in contrast with RPC-over-RDMA version 1, which limits the common portion of all header types to the four words in `rpcrdma_common`.

These changes are part of a larger structural change in the XDR description of RPC-over-RDMA version 2 that enables a cleaner treatment of protocol extension. The XDR appearing in [Section 6](#)

reflects these changes, which are discussed in further detail in [Section 10.1](#).

[3.2](#). `rpcrdma2_hdr_prefix`: Version 2 Transport Header Prefix

The following prefix structure appears at the start of any RPC-over-RDMA version 2 transport header.

<CODE BEGINS>

```
const RPCRDMA2_F_RESPONSE          0x00000001;

struct rpcrdma2_hdr_prefix
    struct rpcrdma_common          rdma_start;
    uint32                          rdma_flags;
```

```
};
```

<CODE ENDS>

The `rdma_flags` is new to RPC-over-RDMA version 2. Currently, the only flag defined within this word is the `RPCRDMA2_F_RESPONSE` flag. The other bits are reserved for future use as described in [Section 9.4](#). The sender MUST set these to zero.

The `RPCRDMA2_F_RESPONSE` flag qualifies the values contained in the transport header's `rdma_start.rdma_xid` and `rdma_start.rdma_credits` fields. The `RPCRDMA2_F_RESPONSE` flag enables a receiver to reliably avoid performing an XID lookup on incoming reverse direction Call messages, and apply the value of the `rdma_start.rdma_credits` field correctly, based on the direction of the message being conveyed.

In general, when a message carries an XID that was generated by the message's receiver (that is, the receiver is acting as a requester), the message's sender sets the `RPCRDMA2_F_RESPONSE` flag. Otherwise that flag is clear. For example:

- o When the `rdma_start.rdma_htype` field has the value `RDMA2_MSG` or `RDMA2_NOMSG`, the value of the `RPCRDMA2_F_RESPONSE` flag MUST be the same as the value of the associated RPC message's `msg_type` field.
- o When the header type is anything else and a whole or partial RPC message payload is present, the value of the `RPCRDMA2_F_RESPONSE` flag MUST be the same as the value of the associated RPC message's `msg_type` field.
- o When no RPC message payload is present, a Requester MUST set the value of `RPCRDMA2_F_RESPONSE` to reflect how the receiver is to

interpret the `rdma_start.rdma_credits` and `rdma_start.rdma_xid` fields.

- o When the `rdma_start.rdma_htype` field has the value `RDMA2_ERROR`, the `RPCRDMA2_F_RESPONSE` flag MUST be set.

[3.3](#). `rpcrdma2_chunk_lists`: Describe External Data Payload

The `rpcrdma2_chunk_lists` structure specifies how an RPC message is conveyed using explicit RDMA operations.

<CODE BEGINS>

```
struct rpcrdma2_chunk_lists {
    uint32                rdma_inv_handle;
    struct rpcrdma2_read_list  *rdma_reads;
    struct rpcrdma2_write_list *rdma_writes;
    struct rpcrdma2_write_chunk *rdma_reply;
};
```

<CODE ENDS>

For the most part this structure parallels its RPC-over-RDMA version 1 equivalent. That is, `rdma_reads`, `rdma_writes`, `rdma_reply` provide, respectively, descriptions of the chunks used to read a long request or directly placed data from the requester, to write directly placed response data into the requester's memory, and to write a long reply into the requester's memory.

An important addition relative to the corresponding RPC-over-RDMA version 1 `rdma_header` structures is the `rdma_inv_handle` field. This field supports remote invalidation of requester memory registrations via the RDMA Send With Invalidate operation.

To request Remote Invalidation, a requester sets the value of the `rdma_inv_handle` field in an RPC Call's transport header to a non-zero value that matches one of the `rdma_handle` fields in that header. If none of the `rdma_handle` values in the header conveying the Call may be invalidated by the responder, the requester sets the RPC Call's `rdma_inv_handle` field to the value zero.

If the responder chooses not to use remote invalidation for this particular RPC Reply, or the RPC Call's `rdma_inv_handle` field contains the value zero, the responder uses RDMA Send to transmit the matching RPC reply.

If a requester has provided a non-zero value in the RPC Call's `rdma_inv_handle` field and the responder chooses to use Remote

Invalidation for the matching RPC Reply, the responder uses RDMA Send

With Invalidate to transmit that RPC reply, and uses the value in the corresponding Call's `rdma_inv_handle` field to construct the Send With Invalidate Work Request.

[4.](#) Transport Properties

RPC-over-RDMA version 2 provides a mechanism for connection endpoints to communicate information about implementation properties, enabling compatible endpoints to optimize data transfer. Initially only a small set of transport properties are defined and a single operation is provided to exchange transport properties (see [Section 5.3.4](#)).

Both the set of transport properties and the operations used to communicate may be extended. Within RPC-over-RDMA version 2, all such extensions are OPTIONAL. For information about existing transport properties, see Sections [4.1](#) through [4.2](#). For discussion of extensions to the set of transport properties, see [Section 9.2](#).

[4.1.](#) Transport Properties Model

A basic set of receiver and sender properties is specified in this document. An extensible approach is used, allowing new properties to be defined in future Standards Track documents.

Such properties are specified using:

- o A code point identifying the particular transport property being specified.
- o A nominally opaque array which contains within it the XDR encoding of the specific property indicated by the associated code point.

The following XDR types are used by operations that deal with transport properties:

<CODE BEGINS>

```
typedef rpcrdma2_propid uint32;

struct rpcrdma2_propval {
    rpcrdma2_propid rdma_which;
    opaque          rdma_data<>;
};

typedef rpcrdma2_propval rpcrdma2_propset<>;

typedef uint32 rpcrdma2_propsubset<>;
```

<CODE ENDS>

An `rpcrdma2_propid` specifies a particular transport property. In order to facilitate XDR extension of the set of properties by concatenating XDR definition files, specific properties are defined as const values rather than as elements in an enum.

An `rpcrdma2_propval` specifies a value of a particular transport property with the particular property identified by `rdma_which`, while the associated value of that property is contained within `rdma_data`.

An `rdma_data` field which is of zero length is interpreted as indicating the default value or the property indicated by `rdma_which`.

While `rdma_data` is defined as opaque within the XDR, the contents are interpreted (except when of length zero) using the XDR typedef associated with the property specified by `rdma_which`. As a result, when `rpcrdma2_propval` does not conform to that typedef, the receiver is REQUIRED to return the error `RDMA2_ERR_BAD_XDR` using the header type `RDMA2_ERROR` as described in [Section 5.3.3](#). For example, the receiver of a message containing a valid `rpcrdma2_propval` returns this error if the length of `rdma_data` is such that it extends beyond the bounds of the message being transferred.

In cases in which the `rpcrdma2_propid` specified by `rdma_which` is understood by the receiver, the receiver also MUST report the error `RDMA2_ERR_BAD_XDR` if either of the following occur:

- o The nominally opaque data within `rdma_data` is not valid when interpreted using the property-associated typedef.
- o The length of `rdma_data` is insufficient to contain the data represented by the property-associated typedef.

Note that no error is to be reported if `rdma_which` is unknown to the receiver. In that case, that `rpcrdma2_propval` is not processed and processing continues using the next `rpcrdma2_propval`, if any.

A `rpcrdma2_propset` specifies a set of transport properties. No particular ordering of the `rpcrdma2_propval` items within it is imposed.

A `rpcrdma2_prosubset` identifies a subset of the properties in a previously specified `rpcrdma2_propset`. Each bit in the mask denotes a particular element in a previously specified `rpcrdma2_propset`. If a particular `rpcrdma2_propval` is at position `N` in the array, then bit number `N mod 32` in word `N div 32` specifies whether that particular `rpcrdma2_propval` is included in the defined subset. Words beyond the last one specified are treated as containing zero.

[4.2.](#) Current Transport Properties

Although the set of transport properties may be extended, a basic set of transport properties is defined in Table 1.

In that table, the columns contain the following information:

- o The column labeled "Property" identifies the transport property described by the current row.
- o The column labeled "Code" specifies the `rpcrdma2_propid` value used to identify this property.
- o The column labeled "XDR type" gives the XDR type of the data used to communicate the value of this property. This data type overlays the data portion of the nominally opaque field `rdma_data` in a `rpcrdma2_propval`.
- o The column labeled "Default" gives the default value for the property which is to be assumed by those who do not receive, or are unable to interpret, information about the actual value of the property.

- o The column labeled "Sec" indicates the section within this document that explains the semantics and use of this transport property.

Property	Cod e	XDR type	Default	Sec
Receive Buffer Size	1	uint32	4096	Section 4.2.1
Reverse Request Support	2	enum rpcrdma2_rv reqsup	RDMA2_RVREQSUP_INLINE	Section 4.2.2

Table 1

[4.2.1.](#) Receive Buffer Size

The Receive Buffer Size specifies the minimum size, in octets, of pre-posted receive buffers. It is the responsibility of the endpoint sending this value to ensure that its pre-posted receive buffers are at least the size specified, allowing the endpoint receiving this value to send messages that are of this size.

<CODE BEGINS>

```
const uint32 RDMA2_PROPID_RBSIZ = 1;
typedef uint32 rpcrdma2_prop_rbsiz;
```

<CODE ENDS>

The sender may use his knowledge of the receiver's buffer size to determine when the message to be sent will fit in the preposted receive buffers that the receiver has set up. In particular,

- o Requesters may use the value to determine when it is necessary to provide a Position-Zero Read chunk when sending a request.
- o Requesters may use the value to determine when it is necessary to provide a Reply chunk when sending a request, based on the maximum possible size of the reply.
- o Responders may use the value to determine when it is necessary, given the actual size of the reply, to actually use a Reply chunk provided by the requester.

[4.2.2.](#) Reverse Request Support

The value of this property is used to indicate a client implementation's readiness to accept and process messages that are part of reverse direction RPC requests.

<CODE BEGINS>

```
enum rpcrdma2_rvreqsup {
    RDMA2_RVREQSUP_NONE    = 0,
    RDMA2_RVREQSUP_INLINE  = 1,
    RDMA2_RVREQSUP_GENL    = 2
};

const uint32 RDMA2_PROPID_BRS = 2;
typedef rpcrdma2_rvreqsup rpcrdma2_prop_brs;
```

<CODE ENDS>

Multiple levels of support are distinguished:

- o The value RDMA2_RVREQSUP_NONE indicates that receipt of reverse direction requests and replies is not supported.
- o The value RDMA2_RVREQSUP_INLINE indicates that receipt of reverse

direction requests or replies is only supported using inline messages and that use of explicit RDMA operations or other form of Direct Data Placement for reverse direction requests or responses is not supported.

- o The value RDMA2_RVREQSUP_GENL that receipt of reverse direction requests or replies is supported in the same ways that forward direction requests or replies typically are.

When information about this property is not provided, the support level of servers can be inferred from the reverse direction requests that they issue, assuming that issuing a request implicitly indicates support for receiving the corresponding reply. On this basis, support for receiving inline replies can be assumed when requests without Read chunks, Write chunks, or Reply chunks are issued, while requests with any of these elements allow the client to assume that general support for reverse direction replies is present on the server.

[5.](#) RPC-over-RDMA Version 2 Transport Messages

[5.1.](#) Overall Transport Message Structure

Each transport message consists of multiple sections:

- o A transport header prefix, as defined in [Section 3.2](#). Among other things, this structure indicates the header type.
- o The transport header proper, as defined by one of the sub-sections below. See [Section 5.2](#) for the mapping between header types and the corresponding header structure.
- o Potentially, an RPC message being conveyed as an addendum to the header.

This organization differs from that presented in the definition of RPC-over-RDMA version 1 [[RFC8166](#)], which presented the first and

second of the items above as a single XDR item. The new organization is more in keeping with RPC-over-RDMA version 2's extensibility model in that new header types can be defined without modifying the existing set of header types.

5.2. Transport Header Types

The new header types within RPC-over-RDMA version 2 are set forth in Table 2. In that table, the columns contain the following information:

- o The column labeled "Operation" specifies the particular operation.
- o The column labeled "Code" specifies the value of header type for this operation.
- o The column labeled "XDR type" gives the XDR type of the data structure used to describe the information in this new message type. This data immediately follows the universal portion on the transport header present in every RPC-over-RDMA transport header.
- o The column labeled "Msg" indicates whether this operation is followed (or not) by an RPC message payload.
- o The column labeled "Sec" indicates the section (within this document) that explains the semantics and use of this operation.

Operation	Code	XDR type	Msg	Sec
Convey Appended RPC Message	0	rpcrdma2_msg	Yes	Section 5.3.1
Convey External RPC Message	1	rpcrdma2_nomsg	No	Section 5.3.2
Report Transport Error	4	rpcrdma2_err	No	Section 5.3.3
Specify Properties at Connection	5	rpcrdma2_connprop	No	Section 5.3.4

+-----+-----+-----+-----+-----+

Table 2

Support for the operations in Table 2 is REQUIRED. Support for additional operations will be OPTIONAL. RPC-over-RDMA version 2 implementations that receive an OPTIONAL operation that is not supported MUST respond with an RDMA2_ERROR message with an error code of RDMA2_ERR_INVALID_HTYPE.

5.3. Header Types Defined in RPC-over-RDMA version 2

The header types defined and used in RPC-over-RDMA version 1 are all carried over into RPC-over-RDMA version 2, although there may be limited changes in the definition of existing header types.

In comparison with the header types of RPC-over-RDMA version 1, the changes can be summarized as follows:

- o To simplify interoperability with RPC-over-RDMA version 1, only the RDMA2_ERROR header (defined in [Section 5.3.3](#)) has an XDR definition that differs from that in RPC-over-RDMA version 1, and its modifications are all compatible extensions.
- o RDMA2_MSG and RDMA2_NOMSG (defined in [Section 5.3.1](#) and [Section 5.3.2](#)) have XDR definitions that match the corresponding RPC-over-RDMA version 1 header types. However, because of the changes to the header prefix, the version 1 and version 2 header types differ in on-the-wire format.
- o RDMA2_CONNPROP (defined in [Section 5.3.4](#)) is a completely new header type devoted to enabling connection peers to exchange information about their transport properties.

5.3.1. RDMA2_MSG: Convey RPC Message Inline

RDMA2_MSG is used to convey an RPC message that immediately follows the Transport Header in the Send buffer. This is either an RPC

request that has no Position-Zero Read chunk or an RPC reply that is not sent using a Reply chunk.

<CODE BEGINS>

```
const rpcrdma2_proc RDMA2_MSG = 0;

struct rpcrdma2_msg {
    struct rpcrdma2_chunk_lists  rdma_chunks;

    /* The rpc message starts here and continues
     * through the end of the transmission. */
    uint32                      rdma_rpc_first_word;
};
```

<CODE ENDS>

[5.3.2.](#) RDMA2_NOMSG: Convey External RPC Message

RDMA2_NOMSG is used to convey an entire RPC message using explicit RDMA operations. Usually this is because the RPC message does not fit within the size limits that result from the receiver's inline threshold. The message may be a Long request, which is read from a memory area specified by a Position-Zero Read chunk; or a Long reply, which is written into a memory area specified by a Reply chunk.

<CODE BEGINS>

```
const rpcrdma2_proc RDMA2_NOMSG = 1;

struct rpcrdma2_nomsg {
    struct rpcrdma2_chunk_lists  rdma_chunks;
};
```

<CODE ENDS>

[5.3.3.](#) RDMA2_ERROR: Report Transport Error

RDMA2_ERROR provides a way of reporting the occurrence of transport errors on a previous transmission. This header type MUST NOT be transmitted by a requester. [cel: how is the XID field set when sending an error report from a requester, or when the error occurred on a non-RPC message?]

<CODE BEGINS>

```
const rpcrdma2_proc RDMA2_ERROR = 4;

struct rpcrdma2_err_vers {
    uint32 rdma_vers_low;
    uint32 rdma_vers_high;
};

struct rpcrdma2_err_write {
    uint32 rdma_chunk_index;
    uint32 rdma_length_needed;
};

union rpcrdma2_error switch (rpcrdma2_errcode rdma_err) {
    case RDMA2_ERR_VERS:
        rpcrdma2_err_vers rdma_vrange;
    case RDMA2_ERR_READ_CHUNKS:
        uint32 rdma_max_chunks;
    case RDMA2_ERR_WRITE_CHUNKS:
        uint32 rdma_max_chunks;
    case RDMA2_ERR_SEGMENTS:
        uint32 rdma_max_segments;
    case RDMA2_ERR_WRITE_RESOURCE:
        rpcrdma2_err_write rdma_writeres;
    case RDMA2_ERR_REPLY_RESOURCE:
        uint32 rdma_length_needed;
    default:
        void;
};
```

<CODE ENDS>

Error reporting is addressed in RPC-over-RDMA version 2 in a fashion similar to RPC-over-RDMA version 1. Several new error codes, and error messages never flow from requester to responder. RPC-over-RDMA version 1 error reporting is described in [Section 5 of \[RFC8166\]](#).

In all cases below, the responder copies the values of the `rdma_start.rdma_xid` and `rdma_start.rdma_vers` fields from the incoming transport header that generated the error to transport header of the error response. The responder sets the `rdma_start.rdma_htype` field of the transport header prefix to `RDMA2_ERROR`, and the `rdma_start.rdma_credit` field is set to the credit grant value for this connection. The receiver of this header type MUST ignore the value of the `rdma_start.rdma_credits` field.

This is the equivalent of ERR_VERS in RPC-over-RDMA version 1. The error code value, semantics, and utilization are the same.

RDMA2_ERR_INVALID_HTYPE

If a responder recognizes the value in the `rdma_start.rdma_vers` field, but it does not recognize the value in the `rdma_start.rdma_htype` field or does not support that header type, it MUST set the `rdma_err` field to RDMA2_ERR_INVALID_HTYPE.

RDMA2_ERR_BAD_XDR

If a responder recognizes the values in the `rdma_start.rdma_vers` and `rdma_start.rdma_proc` fields, but the incoming RPC-over-RDMA transport header cannot be parsed, it MUST set the `rdma_err` field to RDMA2_ERR_BAD_XDR. This includes cases in which a nominally opaque property value field cannot be parsed using the XDR typedef associated with the transport property definition. The error code value of RDMA2_ERR_BAD_XDR is the same as the error code value of ERR_CHUNK in RPC-over-RDMA version 1. The responder MUST NOT process the request in any way except to send an error message.

RDMA2_ERR_READ_CHUNKS

If a requester presents more DDP-eligible arguments than the responder is prepared to Read, the responder MUST set the `rdma_err` field to RDMA2_ERR_READ_CHUNKS, and set the `rdma_max_chunks` field to the maximum number of Read chunks the responder can receive and process.

If the responder implementation cannot handle any Read chunks for a request, it MUST set the `rdma_max_chunks` to zero in this response. The requester SHOULD resend the request using a Position-Zero Read chunk. If this was a request using a Position-Zero Read chunk, the requester MUST terminate the transaction with an error.

RDMA2_ERR_WRITE_CHUNKS

If a requester has constructed an RPC Call message with more DDP-eligible results than the server is prepared to Write, the responder MUST set the `rdma_err` field to RDMA2_ERR_WRITE_CHUNKS, and set the `rdma_max_chunks` field to the maximum number of Write chunks the responder can process and return.

If the responder implementation cannot handle any Write chunks for

a request, it MUST return a response of RDMA2_ERR_REPLY_RESOURCE (below). The requester SHOULD resend the request with no Write chunks and a Reply chunk of appropriate size.

RDMA2_ERR_SEGMENTS

If a requester has constructed an RPC Call message with a chunk that contains more segments than the responder supports, the responder MUST set the `rdma_err` field to RDMA2_ERR_SEGMENTS, and

set the `rdma_max_segments` field to the maximum number of segments the responder can process.

RDMA2_ERR_WRITE_RESOURCE

If a requester has provided a Write chunk that is not large enough to fully convey a DDP-eligible result, the responder MUST set the `rdma_err` field to RDMA2_ERR_WRITE_RESOURCE.

The responder MUST set the `rdma_chunk_index` field to point to the first Write chunk in the transport header that is too short, or to zero to indicate that it was not possible to determine which chunk is too small. Indexing starts at one (1), which represents the first Write chunk. The responder MUST set the `rdma_length_needed` to the number of bytes needed in that chunk in order to convey the result data item.

Upon receipt of this error code, a responder MAY choose to terminate the operation (for instance, if the responder set the index and length fields to zero), or it MAY send the request again using the same XID and more reply resources.

RDMA2_ERR_REPLY_RESOURCE

If an RPC Reply's Payload stream does not fit inline and the requester has not provided a large enough Reply chunk to convey the stream, the responder MUST set the `rdma_err` field to RDMA2_ERR_REPLY_RESOURCE. The responder MUST set the `rdma_length_needed` to the number of Reply chunk bytes needed to convey the reply.

Upon receipt of this error code, a responder MAY choose to terminate the operation (for instance, if the responder set the index and length fields to zero), or it MAY send the request again using the same XID and larger reply resources.

RDMA2_ERR_SYSTEM

If some problem occurs on a responder that does not fit into the above categories, the responder MAY report it to the sender by setting the `rdma_err` field to `RDMA2_ERR_SYSTEM`.

This is a permanent error: a requester that receives this error MUST terminate the RPC transaction associated with the `XID` value in the `rdma_start.rdma_xid` field.

[5.3.4.](#) RDMA2_CONNPROP: Advertise Transport Properties

The `RDMA2_CONNPROP` message type allows an RPC-over-RDMA endpoint, whether client or server, to indicate to its partner relevant transport properties that the partner might need to be aware of.

The message definition for this operation is as follows:

<CODE BEGINS>

```
struct rpcrdma2_connprop {  
    rpcrdma2_propset rdma_props;  
};
```

<CODE ENDS>

All relevant transport properties that the sender is aware of should be included in `rdma_props`. Since support of each of the properties is `OPTIONAL`, the sender cannot assume that the receiver will necessarily take note of these properties. The sender should be prepared for cases in which the receiver continues to assume that the default value for a particular property is still in effect.

Generally, a participant will send a `RDMA2_CONNPROP` message as the first message after a connection is established. Given that fact, the sender should make sure that the message can be received by peers who use the default Receive Buffer Size. The connection's initial receive buffer size is typically 1KB, but it depends on the initial connection state of the RPC-over-RDMA version in use.

Properties not included in `rdma_props` are to be treated by the peer endpoint as having the default value and are not allowed to change

subsequently. The peer should not request changes in such properties.

Those receiving an RDMA2_CONNPROP may encounter properties that they do not support or are unaware of. In such cases, these properties are simply ignored without any error response being generated.

6. XDR Protocol Definition

This section contains a description of the core features of the RPC-over-RDMA version 2 protocol expressed in the XDR language [[RFC4506](#)].

Because of the need to provide for protocol extensibility without modifying an existing XDR definition, this description has some important structural differences from the corresponding XDR description for RPC-over-RDMA version 1, which appears in [[RFC8166](#)].

This description is divided into three parts:

- o A code component license which appears in [Section 6.1](#).

- o An XDR description of the structures that are generally available for use by transport header types including both those defined in this document and those that may be defined as extensions. This includes definitions of the chunk-related structures derived from RPC-over-RDMA version 1, the transport property model introduced in this document, and a definition of the transport header prefixes that precede the various transport header types. This appears in [Section 6.3](#).
- o An XDR description of the transport header types defined in this document, including those derived from RPC-over-RDMA version 1 and those introduced in RPC-over-RDMA version 2. This appears in [Section 6.4](#).

This description is provided in a way that makes it simple to extract into ready-to-compile form. To enable the combination of this description with the descriptions of subsequent extensions to RPC-over-RDMA version 2, the extracted description can be combined with similar descriptions published later, or those descriptions can be

compiled separately. Refer to [Section 6.2](#) for details.

[6.1](#). 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) 2010-2018 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>

[6.2.](#) Extraction and Use of XDR Definitions

The reader can apply the following sed script to this document to produce a machine-readable XDR description of the RPC-over-RDMA version 2 protocol without any OPTIONAL extensions.

<CODE BEGINS>

```
sed -n -e 's:^ */// ::p' -e 's:^ *///$::p'
```

<CODE ENDS>

That is, if this document is in a file called "spec.txt" then the reader can do the following to extract an XDR description file and store it in the file rpcrdma-v2.x.

<CODE BEGINS>

```
sed -n -e 's:^ */// ::p' -e 's:^ *///$::p' \  
    < spec.txt > rpcrdma-v2.x
```

<CODE ENDS>

Although this file is a usable description of the base protocol, when extensions are to supported, it may be desirable to divide into multiple files. The following script can be used for that purpose:

<CODE BEGINS>

```
#!/usr/local/bin/perl
open(IN,"rpcrdma-v2.x");
open(OUT,">temp.x");
while(<IN>)
{
    if (m/FILE ENDS: (.*)$/)
    {
        close(OUT);
        rename("temp.x", $1);
        open(OUT,">temp.x");
    }
    else
    {
        print OUT $_;
    }
}
close(IN);
close(OUT);
```

<CODE ENDS>

Running the above script will result in two files:

- o The file common.x, containing the license plus the common XDR definitions which need to be made available to both the base operations and any subsequent extensions.
- o The file baseops.x containing the XDR definitions for the base operations, defined in this document.

Optional extensions to RPC-over-RDMA version 2, 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.

Alternatively, the XDR descriptions can be compiled separately. In this case the combination of common.x and baseops.x serves to define the base transport, while using as XDR descriptions for extensions, the XDR from the document defining that extension, together with the file common.x, obtained from this document.

6.3. XDR Definition for RPC-over-RDMA Version 2 Core Structures

<CODE BEGINS>

```
/// /*****
/// *   Transport Header Prefixes
/// *****/
///
/// struct rpcrdma_common {
///     uint32      rdma_xid;
///     uint32      rdma_vers;
///     uint32      rdma_credit;
///     uint32      rdma_htype;
/// };
///
/// const RPCRDMA2_F_RESPONSE      0x00000001;
///
/// struct rpcrdma2_hdr_prefix
///     struct rpcrdma_common      rdma_start;
///     uint32                      rdma_flags;
/// };
///
/// /*****
/// *   Chunks and Chunk Lists
/// *****/
///
/// 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 {
///      uint32          rdma_inv_handle;
///      struct rpcrdma2_read_list  *rdma_reads;
///      struct rpcrdma2_write_list *rdma_writes;
///      struct rpcrdma2_write_chunk *rdma_reply;
/// };
///
/// /*****
///  *   Transport Properties
///  *****/
/// /*
///  * Types for transport properties model
///  */
/// typedef rpcrdma2_propid uint32;
///
/// struct rpcrdma2_propval {
///      rpcrdma2_propid rdma_which;
///      opaque          rdma_data<>;
/// };
///
/// typedef rpcrdma2_propval rpcrdma2_propset<>;
/// typedef uint32 rpcrdma2_propsubset<>;
///
/// /*
///  * Transport propid values for basic properties
///  */
/// const uint32 RDMA2_PROPID_RBSIZ = 1;
/// const uint32 RDMA2_PROPID_BRS = 2;
///
/// /*
///  * Types specific to particular properties
///  */

```

```

/// typedef uint32 rpcrdma2_prop_rbsiz;
/// typedef rpcrdma2_rvreqsup rpcrdma2_prop_brs;
///
/// enum rpcrdma2_rvreqsup {
///     RDMA2_RVREQSUP_NONE = 0,
///     RDMA2_RVREQSUP_INLINE = 1,
///     RDMA2_RVREQSUP_GENL = 2
/// };
///
/// /* FILE ENDS: common.x; */

```

<CODE ENDS>

[6.4.](#) XDR Definition for RPC-over-RDMA Version 2 Base Header Types

<CODE BEGINS>

```

/// /******
/// *      Descriptions of RPC-over-RDMA Header Types
/// *      *****/
///
/// /*
/// * Header Type Codes.
/// */
/// const rpcrdma2_proc RDMA2_MSG = 0;
/// const rpcrdma2_proc RDMA2_NOMSG = 1;
/// const rpcrdma2_proc RDMA2_ERROR = 4;
/// const rpcrdma2_proc RDMA2_CONNPROP = 5;
///
/// /*
/// * Header Types to Convey RPC Messages.
/// */
/// struct rpcrdma2_msg {
///     struct rpcrdma2_chunk_lists rdma_chunks;
///
///     /* The rpc message starts here and continues
///      * through the end of the transmission. */
///     uint32 rdma_rpc_first_word;
/// };
///
/// struct rpcrdma2_nomsg {
///     struct rpcrdma2_chunk_lists rdma_chunks;

```

```

/// };
///
/// /*
///  * Header Type to Report Errors.
///  */
/// const uint32 RDMA2_ERR_VERS = 1;
/// const uint32 RDMA2_ERR_BAD_XDR = 2;
/// const uint32 RDMA2_ERR_INVALID_HTYPE = 3;
/// const uint32 RDMA2_ERR_READ_CHUNKS = 4;
/// const uint32 RDMA2_ERR_WRITE_CHUNKS = 5;
/// const uint32 RDMA2_ERR_SEGMENTS = 6;
/// const uint32 RDMA2_ERR_WRITE_RESOURCE = 7;
/// const uint32 RDMA2_ERR_REPLY_RESOURCE = 8;
/// const uint32 RDMA2_ERR_SYSTEM = 9;
///
/// struct rpcrdma2_err_vers {
///     uint32 rdma_vers_low;
///     uint32 rdma_vers_high;
/// };
///

```

```

/// struct rpcrdma2_err_write {
///     uint32 rdma_chunk_index;
///     uint32 rdma_length_needed;
/// };
///
/// union rpcrdma2_error switch (rpcrdma2_errcode rdma_err) {
///     case RDMA2_ERR_VERS:
///         rpcrdma2_err_vers rdma_vrange;
///     case RDMA2_ERR_READ_CHUNKS:
///         uint32 rdma_max_chunks;
///     case RDMA2_ERR_WRITE_CHUNKS:
///         uint32 rdma_max_chunks;
///     case RDMA2_ERR_SEGMENTS:
///         uint32 rdma_max_segments;
///     case RDMA2_ERR_WRITE_RESOURCE:
///         rpcrdma2_err_write rdma_writeres;
///     case RDMA2_ERR_REPLY_RESOURCE:
///         uint32 rdma_length_needed;
///     default:
///         void;
/// };

```

```

///
/// /*
///  * Header Type to Exchange Transport Properties.
///  */
/// struct rpcrdma2_connprop {
///     rpcrdma2_propset rdma_props;
/// };
///
/// /* FILE ENDS: baseops.x; */

```

<CODE ENDS>

6.5. Use of the XDR Description Files

The three files `common.x` and `baseops.x`, when combined with the XDR descriptions for extension defined later, produce a human-readable and compilable description of the RPC-over-RDMA version 2 protocol with the included extensions.

Although this XDR description can be useful in generating code to encode and decode the transport and payload streams, there are elements of the structure of RPC-over-RDMA version 2 which are not expressible within the XDR language as currently defined. This requires implementations that use the output of the XDR processor to provide additional code to bridge the gaps.

- o The values of transport properties are represented within XDR as opaque values. However, the actual structures of each of the properties are represented by XDR typedefs, with the selection of the appropriate typedef described by text in this document. The determination of the appropriate typedef is not specified by XDR, which does not possess the facilities necessary for that determination to be specified in an extensible way.

This is similar to the way in which NFSv4 attributes are handled [[RFC7530](#)] [[RFC5661](#)]. As in that case, implementations that need to encode and decode these nominally opaque entities need to use the protocol description to determine the actual XDR representation that underlays the items described as opaque.

- o The transport stream is not represented as a single XDR object. Instead, the header prefix is described by one XDR object while the rest of the header is described as another XDR object with the mapping between the header type in the header prefix and the XDR object representing the header type represented by tables contained in this document, with additional mappings being specifiable by a later extension document.

This situation is similar to that in which RPC message headers contain program and procedure numbers, so that the XDR for those request and replies can be used to encode and decode the associated messages without requiring that all be present in a single XDR specification. As in that case, implementations need to use the header specification to select the appropriate XDR-generated code to be used in message processing.

- o The relationship between the transport stream and the payload stream is not specified in the XDR itself, although comments within the XDR text make clear where transported messages, described by their own XDR, need to appear. Such data by its nature is opaque to the transport, although its form differs XDR opaque arrays.

Potential extensions allowing continuation of RPC messages across transport message boundaries will require that message assembly facilities, not specifiable within XDR, also be part of transport implementations.

To summarize, the role of XDR in this specification is more limited than for protocols which are themselves XDR programs, where the totality of the protocol is expressible within the XDR paradigm established for that purpose. This more limited role reflects the fact that XDR lacks facilities to represent the embedding of transported material within the transport framework. In addition,

the need to cleanly accommodate extensions has meant that those using rpcgen in their applications need to take a more active role in providing the facilities that cannot be expressed within XDR.

[7.](#) Protocol Version Negotiation

When an RPC-over-RDMA version 2 client establishes a connection to a

server, its first order of business is to determine the server's highest supported protocol version.

As with RPC-over-RDMA version 1, upon connection establishment a client MUST NOT send more than a single RPC-over-RDMA message at a time until it receives a valid non-error RPC-over-RDMA message from the server that grants client credits.

The second word of each transport header is used to convey the transport protocol version. In the interest of simplicity, we refer to that word as `rdma_vers` even though in the RPC-over-RDMA version 2 XDR definition it is described as `rdma_start.rdma_vers`.

First, the client sends a single valid RPC-over-RDMA message with the value two (2) in the `rdma_vers` field. Because the server might support only RPC-over-RDMA version 1, this initial message can be no larger than the version 1 default inline threshold of 1024 bytes.

[7.1.](#) Server Does Support RPC-over-RDMA Version 2

If the server does support RPC-over-RDMA version 2, it sends RPC-over-RDMA messages back to the client with the value two (2) in the `rdma_vers` field. Both peers may use the default inline threshold value for RPC-over-RDMA version 2 connections (4096 bytes).

[7.2.](#) Server Does Not Support RPC-over-RDMA Version 2

If the server does not support RPC-over-RDMA version 2, it MUST send an RPC-over-RDMA message to the client with the same XID, with `RDMA2_ERROR` in the `rdma_start.rdma_htype` field, and with the error code `RDMA2_ERR_VERS`. This message also reports a range of protocol versions that the server supports. To continue operation, the client selects a protocol version in the range of server-supported versions for subsequent messages on this connection.

If the connection is lost immediately after an `RDMA2_ERROR` / `RDMA2_ERR_VERS` message is received, a client can avoid a possible version negotiation loop when re-establishing another connection by assuming that particular server does not support RPC-over-RDMA version 2. A client can assume the same situation (no server support for RPC-over-RDMA version 2) if the initial negotiation message is

lost or dropped. Once the negotiation exchange is complete, both peers may use the default inline threshold value for the transport protocol version that has been selected.

[7.3.](#) Client Does Not Support RPC-over-RDMA Version 2

If the server supports the RPC-over-RDMA protocol version used in Call messages from a client, it MUST send Replies with the same RPC-over-RDMA protocol version that the client uses to send its Calls. The client MUST NOT change the version during the duration of the connection.

[8.](#) Differences from the RPC-over-RDMA Version 1 Protocol

This section describes the substantive changes made in RPC-over-RDMA version 2, as opposed to the structural changes to enable extensibility, which are discussed in [Section 10.1](#).

[8.1.](#) Transport Properties

RPC-over-RDMA version 2 provides a mechanism for exchanging the transport's operational properties. This mechanism allows connection endpoints to communicate the properties of their implementation at connection setup. The mechanism could be expanded to enable an endpoint to request changes in properties of the other endpoint and to notify peer endpoints of changes to properties that occur during operation. Transport properties are described in [Section 4](#).

[8.2.](#) Credit Management Changes

RPC-over-RDMA transports employ credit-based flow control to ensure that a requester does not emit more RDMA Sends than the responder is prepared to receive. [Section 3.3.1 of \[RFC8166\]](#) explains the purpose and operation of RPC-over-RDMA version 1 credit management in detail.

In the RPC-over-RDMA version 1 design, each RDMA Send from a requester contains an RPC Call with a credit request, and each RDMA Send from a responder contains an RPC Reply with a credit grant. The credit grant implies that enough Receives have been posted on the responder to handle the credit grant minus the number of pending RPC transactions (the number of remaining Receive buffers might be zero).

In other words, each RPC Reply acts as an implicit ACK for a previous RPC Call from the requester, indicating that the responder has posted a Receive to replace the Receive consumed by the requester's RDMA Send. Without an RPC Reply message, the requester has no way to know that the responder is properly prepared for subsequent RPC Calls.

Internet-Draft

RDMA Transport for RPC V2

May 2019

Aside from being a bit of a layering violation, there are basic (but rare) cases where this arrangement is inadequate:

- o When a requester retransmits an RPC Call on the same connection as an earlier RPC Call for the same transaction.
- o When a requester transmits an RPC operation that requires no reply.
- o When more than one RPC-over-RDMA message is needed to complete the transaction (e.g., RDMA_DONE).

Typically, the connection must be replaced in these cases. This resets the credit accounting mechanism but has an undesirable impact on other ongoing RPC transactions on that connection.

Because credit management accompanies each RPC message, there is a strict one-to-one ratio between RDMA Send and RPC message. There are interesting use cases that might be enabled if this relationship were more flexible:

- o RPC-over-RDMA operations which do not carry an RPC message; e.g., control plane operations.
- o A single RDMA Send that conveys more than one RPC message for the purpose of interrupt mitigation.
- o An RPC message that is conveyed via several sequential RDMA Sends to reduce the use of explicit RDMA operations for moderate-sized RPC messages.
- o An RPC transaction that needs multiple exchanges or an odd number of RPC-over-RDMA operations to complete.

Bi-directional RPC operation also introduces an ambiguity. If the RPC-over-RDMA message does not carry an RPC message, then it is not possible to determine whether the sender is a requester or a responder, and thus whether the `rdma_credit` field contains a credit request or a credit grant.

A more sophisticated credit accounting mechanism is provided in RPC-over-RDMA version 2 in an attempt to address some of these shortcomings. This new mechanism is detailed in Section TBD.

[8.3.](#) Inline Threshold Changes

The term "inline threshold" is defined in [Section 3.3.2 of \[RFC8166\]](#). An "inline threshold" value is the largest message size (in octets) that can be conveyed on an RDMA connection using only RDMA Send and Receive. Each connection has two inline threshold values: one for messages flowing from client-to-server (referred to as the "client-to-server inline threshold") and one for messages flowing from server-to-client (referred to as the "server-to-client inline threshold"). Note that [\[RFC8166\]](#) uses somewhat different terminology. This is because it was written with only forward-direction RPC transactions in mind.

A connection's inline thresholds determine when RDMA Read or Write operations are required because the RPC message to be sent cannot be conveyed via a single RDMA Send and Receive pair. When 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 RDMA Read or Write operations.

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 invalidated, fencing it from the responder. Memory registration and invalidation typically have a latency cost that is insignificant compared to data handling costs. When a data payload is small, however, the cost of registering and invalidating the memory where the payload resides becomes a relatively significant part of total RPC latency. Therefore the most efficient operation of RPC-over-RDMA occurs when explicit RDMA Read and Write operations are used for large payloads, and are avoided for small payloads.

When RPC-over-RDMA version 1 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.

With NFS version 4.1 [[RFC5661](#)], the increased size of NFS COMPOUND operations resulted in RPC messages that are on average larger and more complex than previous versions of NFS. With 1024-byte inline thresholds, 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 2 increases the default size of inline thresholds. This also increases the maximum size of reverse-direction RPC messages.

[8.4.](#) Support for Remote Invalidation

An STag that is registered using the FRWR mechanism in a privileged execution context or is registered via a Memory Window in an unprivileged context may be invalidated remotely [[RFC5040](#)]. These mechanisms are available when a requester's RNIC supports MEM_MGT_EXTENSIONS.

For the purposes of this discussion, there are two classes of STags. Dynamically-registered STags are used in a single RPC, then invalidated. Persistently-registered STags live longer than one RPC. They may persist for the life of an RPC-over-RDMA connection, or longer.

An RPC-over-RDMA requester may provide more than one STag in one transport header. It may provide a combination of dynamically- and persistently-registered STags in one RPC message, or any combination of these in a series of RPCs on the same connection. Only dynamically-registered STags using Memory Windows or FRWR (i.e., registered via MEM_MGT_EXTENSIONS) may be invalidated remotely.

There is no transport-level mechanism by which a responder can determine how a requester-provided STag was registered, nor whether it is eligible to be invalidated remotely. A requester that mixes persistently- and dynamically-registered STags in one RPC, or mixes them across RPCs on the same connection, must therefore indicate which handles may be invalidated via a mechanism provided in the Upper Layer Protocol. RPC-over-RDMA version 2 provides such a mechanism.

The RDMA Send With Invalidate operation is used to invalidate an STag on a remote system. It is available only when a responder's RNIC supports MEM_MGT_EXTENSIONS, and must be utilized only when a requester's RNIC supports MEM_MGT_EXTENSIONS (can receive and recognize an IETH).

[8.4.1.](#) Reverse Direction Remote Invalidation

Existing RPC-over-RDMA transport protocol specifications [[RFC8166](#)] [[RFC8167](#)] do not forbid direct data placement in the reverse direction, even though there is currently no Upper Layer Protocol that makes data items in reverse direction operations eligible for direct data placement.

When chunks are present in a reverse direction RPC request, Remote Invalidation allows the responder to trigger invalidation of a requester's STags as part of sending a reply, the same way as is done in the forward direction.

Lever & Noveck

Expires November 7, 2019

[Page 33]

Internet-Draft

RDMA Transport for RPC V2

May 2019

However, in the reverse direction, the server acts as the requester, and the client is the responder. The server's RNIC, therefore, must support receiving an IETH, and the server must have registered the STags with an appropriate registration mechanism.

[8.5.](#) Error Reporting Changes

RPC-over-RDMA version 2 expands the repertoire of errors that may be reported by connection endpoints. This change, which is structured to enable extensibility, allows a peer to report overruns of specific resources and to avoid requester retries when an error is permanent.

[9.](#) Extending the Version 2 Protocol

RPC-over-RDMA version 2 is designed to be extensible in a way that enables the addition of OPTIONAL features that may subsequently be converted to REQUIRED status in a future protocol version. The protocol may be extended by Standards Track documents in a way analogous to that provided for Network File System Version 4 as described in [[RFC8178](#)].

This form of extensibility enables limited extensions to the base

RPC-over-RDMA version 2 protocol presented in this document so that new optional capabilities can be introduced without a protocol version change, while maintaining robust interoperability with existing RPC-over-RDMA version 2 implementations. The design allows extensions to be defined, including the definition of new protocol elements, without requiring modification or recompilation of the existing XDR.

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 2 should contain:

- o An explanation of the purpose and use of each new protocol element added.
- o An XDR description including all of the new protocol elements, and a script to extract it.
- o A description of interactions with existing extensions.

This includes possible requirements of other OPTIONAL features to be present for new protocol elements to work, or that a particular

level of support for an OPTIONAL facility is required 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 2 base protocol combined with that for any selected extensions should provide an adequate human-readable description of the extended protocol.

The base protocol specified in this document may be extended within

RPC-over-RDMA version 2 in two ways:

- o New OPTIONAL transport header types may be introduced by later Standards Track documents. Such transport header types will be documented as described in [Section 9.1](#).
- o New OPTIONAL transport properties may be defined in later Standards Track documents. Such transport properties will be documented as described in [Section 9.2](#).

The following sorts of ancillary protocol elements may be added to the protocol to support the addition of new transport properties and header types.

- o New error codes may be created as described in [Section 9.3](#).
- o New flags to use within the `rdma_flags` field may be created as described in [Section 9.4](#).

New capabilities can be proposed and developed independently of each other, and implementers can choose among them. This makes it straightforward to create and document experimental features and then bring them through the standards process.

[9.1](#). Adding New Header Types to RPC-over-RDMA Version 2

New transport header types are to defined in a manner similar to the way existing ones are described in Sections [Section 5.3.1](#) through [Section 5.3.4](#) Specifically what is needed is:

- o A description of the function and use of the new header type.

- o A complete XDR description of the new header type including a description of the use of all fields within the header.
- o A description of how errors are reported, including the definition of a mechanism for reporting errors when the error is outside the available choices already available in the base protocol or in other existing extensions.

- o An indication of whether a Payload stream must be present, and a description of its contents and how such payload streams are used to construct RPC messages for processing.

In addition, there needs to be additional documentation that is made necessary due to the Optional status of new transport header types.

- o Information about constraints on support for the new header types should be provided. For example, if support for one header type is implied or foreclosed by another one, this needs to be documented.
- o A preferred method by which a sender should determine whether the peer supports a particular header type needs to be provided. While it is always possible for a send a test invocation of a particular header type to see if support is available, when more efficient means are available (e.g. the value of a transport property, this should be noted.

[9.2.](#) Adding New Transport properties to the Protocol

The set of transport properties is designed to be extensible. As a result, once new properties are defined in standards track documents, the operations defined in this document may reference these new transport properties, as well as the ones described in this document.

A standards track document defining a new transport property should include the following information paralleling that provided in this document for the transport properties defined herein.

- o The `rpcrdma2_propid` value used to identify this property.
- o The XDR typedef specifying the form in which the property value is communicated.
- o A description of the transport property that is communicated by the sender of `RDMA2_CONNPROP`.
- o An explanation of how this knowledge could be used by the peer receiving this information.

The definition of transport property structures is such as to make it

easy to assign unique values. There is no requirement that a continuous set of values be used and implementations should not rely on all such values being small integers. A unique value should be selected when the defining document is first published as an internet draft. When the document becomes a standards track document, the working group should ensure that:

- o rpcrdma2_propid values specified in the document do not conflict with those currently assigned or in use by other pending working group documents defining transport properties.
- o rpcrdma2_propid values specified in the document do not conflict with the range reserved for experimental use, as defined in [Section 8.2](#).

Documents defining new properties fall into a number of categories.

- o Those defining new properties and explaining (only) how they affect use of existing message types.
- o Those defining new OPTIONAL message types and new properties applicable to the operation of those new message types.
- o Those defining new OPTIONAL message types and new properties applicable both to new and existing message types.

When additional transport properties are proposed, the review of the associated standards track document should deal with possible security issues raised by those new transport properties.

[9.3](#). Adding New Error Codes to the Protocol

New error codes to be returned when using new header types may be introduced in the same Standards Track document that defines the new header type. [cel: what about adding a new error code that is returned for an existing header type?]

For error codes that do not require that additional error information be returned with them, the existing RDMA_ERR2 header can be used to report the new error. The new error code is set as the value of rdma_err with the result that the default switch arm of the rpcrdma2_error (i.e. void) is selected.

For error codes that do require the return of additional error-related information together with the error, a new header type should be defined for the purpose of returning the error together with

needed additional information. It should be documented just like any other new header type.

When a new header type is sent, the sender needs to be prepared to accept header types necessary to report associated errors.

[9.4.](#) Adding New Header Flags to the Protocol

There are currently thirty-one flags available for later assignment. One possible use for such flags would be in a later protocol version, should that version retain the same general header structure as version 2.

In addition, it is possible to assign unused flags within extensions made to version 2, as long as the following practices are adhered to:

- o Flags should not be added to the flag word in the prefix structure if those flags only apply to a single header type. New flags should only be defined for conditions applying to multiple header types.
- o The document defining the new flag should indicate for which header types the flag value is meaningful and for which header types it is an error to set the flag or to leave it unset.
- o The sender needs to be provided with a means to determine whether the receiver is prepared to receive transport headers with the new flag set. This is most likely to take the form of a transport property together with the definition of suitable defaults to use when that property is not supported. Another possibility is to REQUIRE that receivers supporting a particular header type also support a set of additional flags.

[10.](#) Relationship to other RPC-over-RDMA Versions

[10.1.](#) Relationship to RPC-over-RDMA Version 1

In addition to the substantive protocol changes discussed in [Section 8](#), there are a number of structural XDR changes whose goal is to enable within-version protocol extensibility.

The RPC-over-RDMA version 1 transport header is defined as a single XDR object, with an RPC message proper potentially following it. In RPC-over-RDMA version 2, as described in [Section 5.1](#) there are separate XDR definitions of the transport header prefix (see [Section 3.2](#) which specifies the transport header type to be used, and

the specific transport header, defined within one of the subsections of [Section 5](#)). This is similar to the way that an RPC message

consists of an RPC header (defined in [[RFC5531](#)]) and an RPC request or reply, defined by the Upper Layer protocol being conveyed.

As a new version of the RPC-over-RDMA transport protocol, RPC-over-RDMA version 2 exists within the versioning rules defined in [[RFC8166](#)]. In particular, it maintains the first four words of the protocol header as sent and received, as specified in [Section 4.2 of \[\[RFC8166\]\(#\)\]](#), even though, as explained in [Section 3.1](#) of this document, the XDR definition of those words is structured differently.

Although each of the first four words retains its semantic function, there are important differences of field interpretation, besides the fact that the words have different names and different roles with the XDR construct of they are parts.

- o The first word of the header, previously the `rdma_xid` field, retains the format and function that it had in RPC-over-RDMA version 1. Within RPC-over-RDMA version 2, this word is the `rdma_xid` field of the structure `rdma_start`. However, to accommodate the use of request-response pairing of non-RPC messages and the potential use of message continuation, it cannot be assumed that it will always have the same value it would have had in RPC-over-RDMA version 1. As a result, the contents of this field should not be used without consideration of the associated protocol version identification.
- o The second word of the header, previously the `rdma_vers` field, retains the format and function that it had in RPC-over-RDMA version 1. Within RPC-over-RDMA version 2, this word is the `rdma_vers` field of the structure `rdma_start`. To clearly distinguish version 1 and version 2 messages, senders MUST fill in the correct version (fixed after version negotiation) and receivers MUST check that the content of the `rdma_vers` is correct before using referencing any other header field.
- o The third word of the header, previously the `rdma_credit` field, retains the format and general purpose that it had in RPC-over-RDMA version 1. Within RPC-over-RDMA version 2, this word is the `rdma_credit` field of the structure `rdma_start`. The RPC-over-RDMA

version 2 protocol provides additional mechanisms that determine whether the value contained in this field is a credit request or grant. Also, the way in which credits are accounted for may be different in RPC-over-RDMA version 2.

- o The fourth word of the header, previously the union discriminator field `rdma_proc`, retains its format and general function even though the set of valid values has changed. The value of this field is now considered an unsigned 32-bit integer rather than an

enum. Within RPC-over-RDMA version 2, this word is the `rdma_htype` field of the structure `rdma_start`.

Beyond conforming to the restrictions specified in [\[RFC8166\]](#), RPC-over-RDMA version 2 tightly limits the scope of the changes made in order to ensure interoperability. It makes no major structural changes to the protocol, and all existing transport header types used in version 1 (as defined in [\[RFC8166\]](#)) are retained in version 2. Chunks are expressed using the same on-the-wire format and are used in the same way in both versions.

[10.2](#). Extensibility Beyond RPC-over-RDMA Version 2

Subsequent RPC-over-RDMA versions are free to change the protocol in any way they choose as long as they maintain the first four header words as currently specified by [\[RFC8166\]](#).

Such changes might involve deletion or major re-organization of existing transport headers. However, the need for interoperability between adjacent versions will often limit the scope of changes that can be made in a single version.

In some cases it may prove desirable to transition to a new version by using the extension features described for use with RPC-over-RDMA version 2, by continuing the same basic extension model but allowing header types and properties that were OPTIONAL in one version to become REQUIRED in the subsequent version.

[11](#). Security Considerations

The security considerations for RPC-over-RDMA version 2 are the same as those for RPC-over-RDMA version 1.

11.1. Security Considerations (Transport Properties)

Like other fields that appear in each RPC-over-RDMA header, property information is sent in the clear on the fabric with no integrity protection, making it vulnerable to man-in-the-middle attacks.

For example, if a man-in-the-middle were to change the value of the Receive buffer size or the Requester Remote Invalidation boolean, it could reduce connection performance or trigger loss of connection. Repeated connection loss can impact performance or even prevent a new connection from being established. Recourse is to deploy on a private network or use link-layer encryption.

Lever & Noveck

Expires November 7, 2019

[Page 40]

Internet-Draft

RDMA Transport for RPC V2

May 2019

12. IANA Considerations

This document does not require actions by IANA.

13. References

13.1. 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, <<https://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, <<https://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, <<https://www.rfc-editor.org/info/rfc5531>>.
- [RFC8166] Lever, C., Ed., Simpson, W., and T. Talpey, "Remote Direct Memory Access Transport for Remote Procedure Call Version 1", [RFC 8166](#), DOI 10.17487/RFC8166, June 2017, <<https://www.rfc-editor.org/info/rfc8166>>.

- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in [RFC 2119](#) Key Words", [BCP 14](#), [RFC 8174](#), DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

[13.2](#). Informative References

- [IBARCH] InfiniBand Trade Association, "InfiniBand Architecture Specification Volume 1", Release 1.3, March 2015, <http://www.infinibandta.org/content/pages.php?pg=technology_download>.
- [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, <<https://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, <<https://www.rfc-editor.org/info/rfc5041>>.

Lever & Noveck

Expires November 7, 2019

[Page 41]

Internet-Draft

RDMA Transport for RPC V2

May 2019

- [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, <<https://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, <<https://www.rfc-editor.org/info/rfc5662>>.
- [RFC7530] Haynes, T., Ed. and D. Noveck, Ed., "Network File System (NFS) Version 4 Protocol", [RFC 7530](#), DOI 10.17487/RFC7530, March 2015, <<https://www.rfc-editor.org/info/rfc7530>>.
- [RFC8167] Lever, C., "Bidirectional Remote Procedure Call on RPC-over-RDMA Transports", [RFC 8167](#), DOI 10.17487/RFC8167, June 2017, <<https://www.rfc-editor.org/info/rfc8167>>.

[RFC8178] Noveck, D., "Rules for NFSv4 Extensions and Minor Versions", [RFC 8178](https://www.rfc-editor.org/info/rfc8178), DOI 10.17487/RFC8178, July 2017, <<https://www.rfc-editor.org/info/rfc8178>>.

Acknowledgments

The authors gratefully acknowledge the work of Brent Callaghan and Tom Talpey on the original RPC-over-RDMA version 1 specification ([RFC 5666](#)). The authors also wish to thank Bill Baker, Greg Marsden, and Matt Benjamin for their support of this work.

The XDR extraction conventions were first described by the authors of the NFS version 4.1 XDR specification [[RFC5662](#)]. Herbert van den Bergh suggested the replacement sed script used in this document.

Special thanks go to Transport Area Director Spencer Dawkins, NFSV4 Working Group Chairs Spencer Shepler and Brian Pawłowski, and NFSV4 Working Group Secretary Thomas Haynes for their support.

Authors' Addresses

Charles Lever (editor)
Oracle Corporation
1015 Granger Avenue
Ann Arbor, MI 48104
United States of America

Phone: +1 248 816 6463
Email: chuck.lever@oracle.com

Lever & Noveck

Expires November 7, 2019

[Page 42]

Internet-Draft

RDMA Transport for RPC V2

May 2019

David Noveck
NetApp
1601 Trapelo Road
Waltham, MA 02451
United States of America

Phone: +1 781 572 8038
Email: davenoveck@gmail.com

