

Network File System Version 4
Internet-Draft
Intended status: Informational
Expires: June 8, 2018

D. Noveck
NetApp
December 5, 2017

RPC-over-RDMA Extensions to Reduce Internode Round-trips
draft-dnoveck-nfsv4-rpcrdma-rtrext-03

Abstract

It is expected that a future version of the RPC-over-RDMA transport will allow protocol extensions to be defined. This would provide for the specification of OPTIONAL features allowing participants who implement such features to cooperate as specified by that extension, while still interoperating with participants who do not support that extension.

A particular extension is described herein, whose motivation is to reduce the latency due to inter-node round-trips needed to effect operations which involve direct data placement or which transfer RPC messages longer than the fixed inline buffer size limit.

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 June 8, 2018.

Copyright Notice

Copyright (c) 2017 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

Internet-Draft

RPC/RDMA Round-trip Reductions

December 2017

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Preliminaries	3
1.1.	Requirements Language	3
1.2.	Introduction	3
1.3.	Role of this Document	3
1.4.	Prerequisites	4
1.5.	Participant Terminology	5
2.	Extension Overview	5
3.	Data Placement Features	6
3.1.	Current Situation	6
3.2.	RDMA_MSGP	6
3.3.	Send-based Data Placement	8
3.4.	Other Extensions Relating to Data Placement	8
4.	Message Continuation Feature	9
4.1.	Current Situation	9
4.2.	Message Continuation Changes	10
4.3.	Message Continuation and Credits	10
5.	Using Protocol Additions	11
5.1.	New Operation Support	12
5.2.	Message Continuation Support	12
5.3.	Support for Send-based Data Placement	13
5.4.	Error Reporting	14
6.	XDR Preliminaries	15
6.1.	Message Continuation Preliminaries	15
6.2.	Data Placement Preliminaries	15
7.	Data Placement Structures	18
7.1.	Data Placement Overview	18
7.2.	Buffer Structure Definition	19
7.3.	Message Data Placement Structures	21
7.4.	Response Direction Data Placement Structures	23
8.	Transport Properties	25
8.1.	Property List	25
8.2.	RTR Support Property	26
8.3.	Receive Buffer Structure Property	26
8.4.	Request Transmission Receive Limit Property	27

8.5.	Response Transmission Send Limit Property	27
9.	New Operations	28
9.1.	Operations List	28
9.2.	Transmit Request Operation	29
9.3.	Transmit Response Operation	29

9.4.	Transmit Continue Operation	30
9.5.	Error Reporting Operation	31
10.	XDR	34
10.1.	Code Component License	35
10.2.	XDR Proper for Extension	37
11.	Security Considerations	42
12.	IANA Considerations	43
13.	References	43
13.1.	Normative References	43
13.2.	Informative References	43
	Acknowledgments	44
	Author's Address	44

[1.](#) Preliminaries

[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](#)].

[1.2.](#) Introduction

This document describes a potential extension to the RPC-over-RDMA protocol, which would allow participating implementations to have more flexibility in how they use RDMA sends and receives to effect necessary transmission of RPC requests and replies.

In contrast to existing facilities defined in RPC-over-RDMA Version One in which the mapping between RPC messages and RPC-over-RDMA messages is strictly one-to-one and placement of bulk data is effected only through use of explicit RDMA operations, the following features are made available through this extension:

- o The ability to effect data placement in the context of a single RPC-over-RDMA transmission, rather than requiring explicit RDMA

operations to effect the necessary placement.

- o The ability to continue an RPC request or reply over multiple RPC-over-RDMA transmissions.

1.3. Role of this Document

This is not a standards-track document, despite the fact that it contains many of the sorts of items (e.g. proposed XDR, detailed field descriptions) that normally appear in such documents.

Noveck

Expires June 8, 2018

[Page 3]

Internet-Draft

RPC/RDMA Round-trip Reductions

December 2017

Although this document is in the informational category it is not expected to result in an Informational RFC, as the material within it is not expected to be of interest to the internet community in general. Its target audience is the nfsv4 working group itself and it is not expected to evolve into an RFC.

The function of this document is essentially exploratory, in that it looks at a number of possible ways that the RPC-over-RDMA transport could be extended. Although many of these might well be followed up on eventually with standards-track documents, it should not be assumed that all will or that the relation among the various elements of any extension to address these issue will be the same as laid out here.

1.4. Prerequisites

This document is written assuming that certain underlying facilities will be made available to build upon, in the context of a future version of RPC-over-RDMA. It is most likely that such facilities will be first available in Version Two of RPC-over-RDMA.

- o A protocol extension mechanism is needed to enable the extensions to RPC-over-RDMA described here.

This document is currently written to conform to the extension model for the proposed RPC-over-RDMA Version Two as described in [[I-D.cel-nfsv4-rpcrdma-version-two](#)].

- o An existing means of communicating transport properties between

the RPC-over-RDMA endpoints is assumed.

This document is currently written assuming the transport property model defined in [[I-D.cel-nfsv4-rpcrdma-version-two](#)]. will be available and can be extended to meet the needs of this extension.

As the document referred to above is currently a personal Internet Draft, and subject to change, adjustments to this document are expected to be necessary when and if the needed facilities are defined in one or more working group documents leading to the potential publication of Standards-track RFCs.

Such an RFC for a new RPC-over-RDMA version might differ from [[I-D.cel-nfsv4-rpcrdma-version-two](#)] in significant ways even if it provides the prerequisites listed above. For example,

- o The extension model might be significantly different. For example, it might use an approach more like that used in [[RFC8178](#)]

rather than using a single message type as a vehicle for OPTIONAL extensions.

- o There is the possibility of significant change in the credit model. While [[I-D.cel-nfsv4-rpcrdma-version-two](#)] contains support for one-way messages, much of the text regarding credits is inherited from [[RFC8166](#)] which assumes a one-to-one mapping between requests and responses. It is not clear whether this mismatch will be resolved by changing (only) the description of the credit mechanism or whether a more basic protocol change is required. Whichever approach is taken, the treatment of message continuation is likely to follow.

[1.5.](#) Participant Terminology

A number of different terms are used regarding the roles of the two participants in an RPC-over-RMA connection. Some of these roles last for the duration of a connection while others vary from request to request or from message to message.

The roles of the client and server are fixed for the lifetime of the connection, with the client defined as the endpoint which initiated

the connection.

The roles of requester and responder often parallel those of client and server, although this is not always the case. Most requests are made in the forward direction, in which the client is the requester and the server is the responder. However, backward direction requests are possible, in which case the server is the requester and the client is the responder. As a result clients and servers may both act as requesters and responders for different requests issued on the same connection.

The roles of sender and receiver vary from message to messages. With regard to the messages described in this document, the sender may act as a requester by sending RPC requests or a responder by sending RPC requests or as both at the same time by sending a mix of the two.

[2.](#) Extension Overview

This extension is intended to function as part of RPC-over-RDMA and implementations should successfully interoperate with existing RPC-over-RDMA Version One implementations. Nevertheless, this extension seeks to take a somewhat different approach to high-performance RPC operation than has been used previously in that it seeks to de-emphasize the use of explicit RDMA operations. It does this in two ways:

- o By implementing a send-based form of data placement (see [Section 3](#)), use of explicit RDMA operations can be avoided in many common cases in which data is to be placed at an appropriate location in the receiver's memory.
- o Use of explicit RDMA to support reply chunks and position-zero read chunks can be avoided by allowing a single message to be split into multiple transmissions. This can be used to avoid many instances of the only existing use of explicit RDMA operations not associated with Direct Data Placement.

While use of explicit RDMA operations allows the cost of the actual data transfer to be offloaded from the client and server CPUs to the RNIC, there are ancillary costs in setting up the transfer that cannot be ignored. As a result, send-based functions are often

preferable, since the RNIC also uses DMA to effect these operations. In addition, the cost of the additional inter-node round trips required by explicit RDMA operation can be an issue, which can become increasingly troublesome as internode distances increase. Once one moves from in-machine-room to campus-wide or metropolitan-area distances the additional round-trip delay of 16 microseconds per mile becomes an issue impeding use of explicit RDMA operations.

[3.](#) Data Placement Features

[3.1.](#) Current Situation

Although explicit RDMA operations are used in the existing RPC-over-RDMA protocol for purposes unrelated to Direct Data Placement, all placement of bulk data is effected using explicit RDMA operations.

As a result, many operations requiring placement of bulk data involve multiple internode round trips.

[3.2.](#) RDMA_MSGP

Although this was not stated explicitly, it appears that RDMA_MSGP (defined in [[RFC5666](#)], removed from RPC-over-RDMA Version One by [[RFC8166](#)]), was an early attempt to effect correct placement of bulk data within a single RPC-over-RDMA transmission.

As things turned out, the fields within the RDMA_MSGP header were not described in [[RFC5666](#)] in a way that allowed this message type to be implemented.

In attempting to provide the appropriate data placement functionality, we have to keep in mind and avoid the problems that led to failure of RDMA_MSGP. It appears that the problems go deeper

than neglecting to write a few relevant sentences. It is helpful to note that:

- o The inline message size limits eventually adopted were too small to allow RDMA_MSGP to be used effectively. This is true of both the 1K limit in Version One [[RFC8166](#)] and the 4K limit specified in [[I-D.cel-nfsv4-rpcrdma-version-two](#)].

On the other hand, there is text within [[RFC5667](#)] that suggests that much longer messages were anticipated at some points during the evolution of RPC-over-RDMA.

- o The fact that NFSv4 COMPOUNDS often have additional operations beyond the one including the bulk data means that the RDMA_MSGP model cannot be extended to NFSv4. As a result, the bulk data needs to be excised from the data stream just as chunks are, so that the payload stream can include non-bulk data both before and after the logical position of the excised bulk data.
- o In order for the sender to determine the appropriate amount of padding necessary within a transmission to place the bulk data at the proper position within receive buffer, the server must know more about the structure of the receiver's buffers. Since the padding needs to bring the bulk data to a position within the buffer that is appropriate to receive the bulk data, the sender needs to know where within the receive buffers such placement-eligible areas are located.
- o While appropriate padding could place the bulk data within a large WRITE into an appropriately aligned buffer or set of buffer, there is no corresponding provision for the bulk data associated with a READ. In short, there is no way to indicate to the responder that it should use RDMA_MSGP to appropriately place bulk data in the response.
- o There is no explicit discussion of the required padding's use in effecting proper data placement or connection with the ULB's specification of DDP-eligible XDR items.

To summarize, RDMA_MSGP was an attempt to properly place bulk data which was thought of as a local optimization and insufficient attention was given to it to make it successful. As a result, as RPC-over-RDMA Version One was developed, data placement was identified with the use of explicit RDMA operations providing DDP and the possibility of dataplacement within sends was not recognized.

In this extension we will describe a more complete way to provide send-based data placement, as follows:

- o By defining the structure of receive buffers as a transport property available to be interrogated by the peer implementation.
- o By treating positioning of bulk data within a message as an instance of data placement, causing the bulk data to be excised from the payload XDR stream, as is the case with other forms of bulk data placement (e.g. DDP).
- o By defining new data structures to control placement of bulk data that support both send-based data placement and DDP using explicit RDMA operations that was an integral part in RPC-over-RDMA Version One. These new control structures, described in [Section 7.1](#) are organized differently from the chunk-based structures described in [[RFC8166](#)].

In making these changes, we will retain certain aspects of the DDP model:

- o The set of bulk data items eligible for special data placement is exactly the same as with DDP, as defined by the RPC protocol's upper-layer binding document.
- o The concept of an inline XDR stream is retained, with specially placed items appearing outside it, but with references to them retained so that the receiver has access to all of the message data.

[3.4.](#) Other Extensions Relating to Data Placement

In order to support send-based data placement, new placement-related data structures have been defined, as described in Sections [7.3](#) and [7.4](#).

These new data structures support both send-based and RDMA-operation-based data placement. In addition, because of the restructuring described in [Section 7.1](#), a number of additional facilities are made available:

- o The ability to restrict entries regarding data placement in response data to XDR data items generated in response to performing particular constituent operations within a given RPC request (e.g. specific operations within an NFSv4 COMPOUND).

- o The ability to make use of special data placement contingent on the actual length of a placement-eligible data item in the response.
- o The ability to specify whether use of data placement for a particular placement-eligible data item is required or optional.

These additional facilities will be available to implementations that do not support send-based data placement, as long as both parties support the OPTIONAL Header types that include these new structures. For more information about the relationships among, the new transport properties, operations, and features, see [Section 5](#).

[4.](#) Message Continuation Feature

[4.1.](#) Current Situation

Within RPC-over-RDMA Version One [[RFC8166](#)], each transmission of a request or reply involves sending a single RDMA send message and conversely each message-related transmission involves only a single RPC request or reply.

This strict one-to-one model leads to some potential performance issues.

- o Because of RDMA's use of fixed-size receives, some requests and replies will inevitably not fit in the limited space available, even if they do not contain any DDP-eligible bulk data.

Such cases will raise performance issues because, to deal with them, the server is interrupted twice to receive a single request and all the necessary transfers are serialized. In particular, there are two server interrupt latencies involved before the server can process the actual request, in addition to the OTW round-trip latencies.

- o In the case of replies, there may be cases in which reply chunks need to be allocated and registered even if the actual reply would fit within the fixed receive-size limit. Because the decision to create a reply chunk is made at the time the request is sent, even an extremely low probability of a longer reply will trigger allocation of a reply chunk.

Because this decision is made in conformance with ULB rules, which, by their nature, may only reference a limited set of data, a reply chunk may be required even when the actual probability of

a long reply is exactly zero. For example a GETATTR request can generate a long reply due to a long ACL, and thus COMPOUND with

this operation might allocate a reply chunk, even if the specific file system being interrogated only supports ACLs of limited sizes, or the GETATTR in question does not interrogate one of the ACL attributes. Also, the OWNER attribute is a string and it may be impossible to determine a priori that the owner of any particular file has no chance of requiring more than 4K bytes of space, for example. The assumption that there are no such user names, while it probably is valid, is not a fact that RPC-over-RDMA implementations can depend on.

[4.2.](#) Message Continuation Changes

Continuing a single RPC request or reply is addressed by defining separate optional header types to begin and to continue sending a single RPC message. This is instead of creating a header with a continuation bit. In this approach, all of the fields relating to data placement, which include support for send-based data placement, appear in the starting header (of types ROPT_XMTREQ and ROPT_XMTRESP) and apply to the RPC message as a whole.

Later RPC-over-RDMA messages (of type ROPT_XMTCONT) may extend the payload stream and/or provide additional buffers to which bulk data can be directed.

In this case, all of the RPC-over-RDMA messages used together are referred to as a transmission group and must be received in order without any intervening message.

In implementations using this optional facility, those decoding RPC messages received using RPC-over-RDMA no longer have the assurance that that each RPC message is in a contiguous buffer. As most XDR implementations are built based on the assumption that input will not be contiguous, this will not affect performance in most cases.

[4.3.](#) Message Continuation and Credits

Using multiple transmissions to send a single request or response can complicate credit management. In the case of the message continuation feature, deadlocks can be avoided because use of message

continuation is not obligatory. The requester or responder can use explicit RDMA operations if sufficient credits to use message continuation are not available.

A requester is well positioned to make this choice with regard to the sending of requests. The requester must know, before sending a request, how long it will be, and therefore, how many credits it would require to send the request using message continuation. If these are not available, it can avoid message continuation by either

creating read chunks sufficient to make the payload stream fit in a single transmission or by creating a position-zero read chunk.

With regard to the response, the requester is not in position to know exactly how long the response will be. However, the ULB will allow the maximum response length to be determined based on the request. This value can be used:

- o To determine the maximum number of receive buffers that might be required to receive any response sent.
- o To allocate and register a reply chunk to hold a possible large reply.

The requester can avoid doing the second of these if the responder has indicated it can use message continuation to send the response. In this case, it makes sure that the buffers will be available and indicates to the responder how many additional buffers (in the form of pre-posted reads have been made available to accommodate continuation transmissions.

When the responder processes the request, those additional receive buffers may be used or not, or used only in part. This may be because the response is shorter than the maximum possible response, or because a reply chunk was used to transmit the response.

After the first or only transmission associated with the response is received by the requester, it can be determined how many of the additional buffers were used for the response. Any unused buffers can be made available for other uses such as expanding the pool of receive buffers available for the initial transmissions of response or for receiving opposite direction requests. Alternatively, they

can be kept in reserve for future uses, such as being made available to future requests which have potentially long responses.

[5.](#) Using Protocol Additions

In using existing RPC-over-RDMA facilities for protocol extension, interoperability with existing implementations needs to be assured. Because this document describes support for multiple features, we need to clearly specify the various possible extensions and how peers can determine whether certain facilities are supported by both ends of the connection.

[5.1.](#) New Operation Support

Note that most of the new operations defined in this extension are not tightly tied to a specific feature. `XOPT_XMTREQ` and `XOPT_XMTRESP` are designed to support implementations that support either or both Send-based data placement or message continuation. However, the converse is not the case and these header types can be implemented by those not supporting either of these features. For example, implementations may only need support for the facilities described in [Section 3.4](#).

Implementations may determine whether a peer implementation supports `XOPT_XMTREQ`, `XOPT_XMTREQ`, or `XOPT_XMTCONT` by attempting these operations. An alternative is to interrogate the RTR Support Property for information about which operations are supported.

[5.2.](#) Message Continuation Support

Implementations may determine and act based on the level of peer implementation of support for message continuation as follows:

- o To deal with issues relating to sending the peer multi-transmission requests, the requester can interrogate the peer's value of the Request Transmission Receive Limit ([Section 8.4](#)). In cases in which the property is not provided or has the value one,

the requester implementation can avoid sending multi-transmission requests, and use the equivalent of position-zero read chunks to convey a request larger than the receive buffer limit.

Similarly, if the request is longer than can fit in a set of transmissions given that limit, the request can be conveyed in the same fashion,

- o To deal with issues relating to sending the peer multi-transmission responses, responders will only send multi-transmission responses for requests conveyed using XOPT_XMTREQ where the number of response transmissions is less than or equal to buffer reservation count (in the field optxrq_rsbuf). The requester can avoid receiving a message consisting of too many transmissions by setting this field appropriately. This includes the case in which the requester cannot handle any multi-transmission responses.
- o To avoid reserving receive buffers that the responder is not prepared to use, the requester can interrogate the peer's value of the Response Transmission Send Receive Limit ([Section 8.5](#)). In cases in which it is possible that a request might result in a response too large for this set of buffers, the requester, the

requester can provide a reply chunk to receive the response, which the responder can use if the count of buffers provided is insufficient.

[5.3](#). Support for Send-based Data Placement

Implementations may determine and adapt to the level of peer implementation support for send-based data placement as described below. Note that an implementation may be able to send messages containing bulk data items placed using send-based data placement while not being prepared to receive them, or the reverse.

- o The requester can interrogate the responder's Receive Buffer Structure Property. In cases in which the property is not provided or shows no placement-targetable buffer segments, an implementation knows that messages containing bulk data may not be sent using send-based data placement. In such cases, when XOPT_XMTREQ is used to send a request, bulk items may be

transferred by setting the associated placement information to indicate that the bulk data is to be fetched using explicit RDMA operations.

- o In cases in which a requester is unprepared to accept messages using send-based data placement, its Receive Buffer Structure Property will make this clear to the responder. Nevertheless, the requester will generally indicate to the responder that bulk data items are to be returned using explicit RDMA operations. As a result, requesters may use XOPT_XMTREQ (and get the benefit of the placement-related features discussed in [Section 3.4](#) even if they support neither message continuation nor send-based data placement.
- o Since it is possible for a responder to generate responses containing bulk data using send-based data placement even if it is not prepared to send such message, a requester who is prepared to accept such messages should specify in the request that the responses are to contain (or may contain) bulk data placed in this way. In deciding whether this is to be done the requester can interrogate the responder's RTR Support Property for information about which whether the peer can send responses in this form. It can do this without regard to whether the responder can accept messages containing bulk data items placed using send-based data placement.

In determining whether bulk data will be placed using send-based data placement or via explicit RDMA operations, the level of support for message continuation will have a role. This is because DDP using explicit RDMA will reduce message size while send-based data

placement reduces the size of the payload stream by rearranging the message, leaving the message size the same. As a result, the considerations discussed in [Section 4.3](#) will have to be attended to by the sender in determining which form of data placement is to be used.

[5.4](#). Error Reporting

The more extensive transport layer functionality described in this document requires its own means of reporting errors, to deal with issues that are distinct from:

- o Errors (including XDR errors) in the XDR stream as received by responder or requester.
- o XDR errors detected in the XDR headers defined by the base protocol.
- o XDR errors detected in the new operations defined in this document.

Beyond the above, the following sorts of errors will have to be dealt with, depending on which of the features of the extension are implemented.

- o Information associated with send-based data placement may be inconsistent or otherwise invalid, even though it conforms to the XDR definition.
- o There may be problems with the organization of transmission groups in that there are missing or extraneous transmissions.

In each of the above cases, the problem will be reported to the sender using the Error Reporting operation which needs to be supported by every endpoint that sends ROPT_XMTREQ, ROPT_XMTRESP, or ROPT_XMTCONT. This includes cases in which the problem is one with a reply. The function of the Error Reporting operation is to aid in diagnosing transport protocol errors and allowing the sender to recover or decide recovery is not possible. Reporting failure to the requesting process is dealt with indirectly. For example,

- o When the transmissions used to send a request are ill-formed, the requestor can respond to the error indication by proceeding to send the request using existing (i.e. non-extended) facilities. If it chooses not to do so, the requestor can report an RPC request failure to the initiator of the RPC.

- o When the transmissions used to send a response are ill-formed, the responder need to know about the problem since it will otherwise assume that the transmissions succeeded. It can proceed to resend the reply using existing (i.e. non-extended) facilities. If it

chooses not to do so, the requester will not see a response and eventually an RPC timeout will occur.

[6.](#) XDR Preliminaries

[6.1.](#) Message Continuation Preliminaries

In order to implement message continuation, we have occasion to refer to particular RPC-over-RDMA transmissions within a transmission group or to characteristics of a later transmission group.

<CODE BEGINS>

```
typedef uint32  xms_grpxn;
typedef uint32  xms_grpxc;
struct xms_id {
    uint32      xmsi_xid;
    msg_type     xmsi_dir;
    xms_grpxn    xmsi_seq;
}
```

<CODE ENDS>

An `xms_grpxn` designates a particular RPC-over-RDMA transmission within a set of transmissions devoted to sending a single RPC message.

An `xms_grpxc` specifies the number of RPC-over-RDMA transmissions in a potential group of transmissions devoted to sending a single RPC message.

[6.2.](#) Data Placement Preliminaries

Data structures related to data placement use a number of XDR typedefs to help clarify the meaning of fields in the data structures which use these typedefs.

<CODE BEGINS>

```
typedef uint32  xmdp_itemlen;
typedef uint32  xmdp_pldisp;
typedef uint32  xmdp_vsdisp;

typedef uint32  xmdp_tbsn;

enum xmdp_type {
    XMPTYPE_EXRW = 1,
    XMPTYPE_TBSN = 2,
    XMPTYPE_CHOOSE = 3,
    XMPTYPE_BYSIZE = 4,
    XMPTYPE_TOOSHORT = 5,
    XMPTYPE_NOITEM = 6
};
```

<CODE ENDS>

An `xmdp_itemlen` specifies the length of XDR item. Because items excised from the XDR stream are XDR items, lengths of items excised from the XDR stream are denoted by `xmdp_itemlens`.

An `xmdp_pldisp` specifies a specific displacement with the payload stream associated with a single RPC-over-RDMA transmission or a group of such transmissions. Note that when multiple transmissions are used for a single message, all of the payload streams within a transmission group are considered concatenated.

An `xmdp_vsdisp` specifies a displacement within the virtual XDR stream associates with the set of RPC messages transferred by single RPC-over-RDMA transmission or a group of such transmissions. The virtual XDR stream includes bulk data excised from the payload stream and so displacements within it reflect those of the corresponding objects in the XDR stream that might be sent and received if no bulk data excision facilities were involved in the RPC transmission.

An `xmdp_tbsn` designates a particular target buffer segment within a (trivial or non-trivial) RPC-over-RDMA transmission group. Each placement-targetable buffer segment is assigned a number starting with zero and proceeding through all the buffer segments for all the RPC-over-RDMA transmissions in the group. This includes buffer segments not actually used because transmission are shorter than the maximum size and those in which a placement-targetable buffer segment is used to hold part of the payload XDR stream rather than bulk data.

An `xmdp_type` allows a selection between placement using explicit RDMA

operations (i.e. DDP) and send-based data placement. Fields of this

type are used in a number of contexts. The specific context governs which subset of the types is valid:

- o In request messages, they indicate where each of the specially placed data items within the request has been placed. In this case, `xmdp_type` appears as the discriminator within an `xmdp_loc` which is part of an `xmdp_mitem` that is an element within a request's `optxrq_dp` field.
- o In request messages, they direct the responder as to where potential specially placed items are to be placed. In this case, `xmdp_type` appears as the discriminator within an `xmdp_rsdloc` which is part of an `xmdp_rsditem` that is an element within a request's `optxrq_rsd` field.
- o In response messages, they indicate how each of the potential specially placed items has been dealt with. A subset of these specially placed data items and are presented in the same form as that used for specially placed data items within a request. In this case, `xmdp_type` appears as the discriminator within an `xmdp_loc` which is part of an `xmdp_mitem` that is an element within a response's `optxrs_dp` field.

A number of these type are valid in all of these contexts, since they specify use of a specific mode of data placement which is to be used or has been used.

- o `XMPTYPE_EXRW` selects DDP using explicit RDMA reads and writes.
- o `XMPTYPE_TBSN` selects use of send-based data placement in which placement-eligible data is located in placement-targetable buffer segments.

Another set of types is used to direct the use of specific sets of types but cannot specify an actual choice that has been made.

- o `XMPTYPE_CHOICE` indicates that the responder may use either send-based data placement or chunk-based DDP using explicit RDMA operations, with a target location for the latter having been provided by the requester.

- o XMPTYPE_BYSIZE indicates that the responder is to use either send-based data placement or chunk-based DDP using explicit RDMA operations, with the choice between the two governed by the actual size of the associated DDP-eligible XDR item.

The following types are used when no actual special placement has occurred. They are used in responses to indicate ways in which a

direction to govern data placement in a reply was responded to without resulting in special placement.

- o XMPTYPE_TOOSHORT indicates that the corresponding entry in an xmdp_rsdset was matched with a DDP-eligible item which was too small to be handled using special placement, resulting in the DDP-eligible item being placed inline.
- o XMPTYPE_NOITEM indicates that the corresponding entry in an xmdp_rsdset was not matched with a DDP-eligible item in the reply.

The following table indicates which of the above types is valid in each of the contexts in which these types may appear. For valid occurrences, it distinguishes those which give sender-generated information about the message, and those that direct reply construction, from those that indicate how those directions governed the construction of a reply. For invalid occurrences, we distinguish between those that result in XDR decode errors and those which are valid from the XDR point of view but are semantically invalid.

Type	xmdp_loc in request	xmdp_rsdloc in request	xmdp_loc in response
XMPTYPE_EXRW	Valid Info	Valid Direction	Valid Result
XMPTYPE_TBSN	Valid Info	Valid Direction	Valid Result
XMPTYPE_BYSIZE	XDR Invalid	Valid Direction	XDR Invalid
XMPTYPE_CHOICE	XDR Invalid	Valid Direction	XDR Invalid
XMPTYPE_TOOSHORT	Sem. Invalid	XDR Invalid	Valid Result
XMPTYPE_NOITEM	Sem. Invalid	XDR Invalid	Valid Result

Table 1

[7.](#) Data Placement Structures

[7.1.](#) Data Placement Overview

To understand the new data placement structures defined here, it is necessary to review the existing DDP structures used in RPC-over-RDMA Version One and look at the corresponding structures in the new message transmission headers defined in this document.

We look first at the existing structures.

- o Read chunks are specified on requests to indicate data items to be excised from the payload stream and fetched from the requester's

memory by the responder. As such, they serve as a means of supplying data excised from the payload XDR stream.

Read chunks appear in replies but they have no clear function there.

- o Write chunks are specified on requests to provide locations in requester memory to which DDP-eligible items in the corresponding reply are to be transferred. They do not describe data in the request but serve to direct reply construction.

When write chunks appear in replies they serve to indicate the length of the data transferred. The addresses to which the bulk reply data has been transferred is available, but this information is already known to the requester.

- o Reply chunks are specified to provide a location in the requester's memory to which the responder can transfer the response using RDMA Write. Like write chunks, they do not describe data in the request but serve to direct reply construction.

When reply chunks appear in reply message headers, they serve mainly to indicate whether the reply chunk was actually used.

Within the data placement structures defined here a different

organization is used, even where DDP using explicit RDMA operations is supported.

- o All messages that contain bulk data contain structures that indicate where the excised data is located. See [Section 7.3](#) for details.
- o Requests that might generate replies containing bulk data contain structures that provide guidance as to where the bulk data is to be placed. See [Section 7.4](#) for details.

Both sets of data structure are defined at the granularity of an RPC-over-RDMA transmission group. That is, they describe the placement of data within an RPC message and the scope of description is not limited to a single RPC-over-RDMA transmission.

[7.2.](#) Buffer Structure Definition

Buffer structure definition information is used to allow the sender to know how receive buffers are constructed, to allow it to appropriately pad messages being sent so that bulk data will be received into a memory area with the appropriate characteristics.

In this case, data placement will not place data in a specific address, picked and registered in advance as is done to effect DDP using explicit RDMA operations. Instead, a message is sent so that when it is matched with one of the preposted receives, the bulk data will be received into a memory area with the appropriate characteristics, including:

- o size
- o alignment
- o placement-targetability and potentially other memory characteristics such as speed, persistence.

<CODE BEGINS>

```
struct xmrbs_seg {
    uint32      xmrseg_length;
    uint32      xmrseg_align;
```

```

        uint32          xmrseg_flags;
};

const uint32    XMRSFLAG_PLT = 0x01;

struct xmrbs_group {
    uint32          xmrgrp_count;
    xmrbs_seg       xmrgrp_info;
};

struct xmrbs_buf {
    uint32          xmrbuf_length;
    xmrbs_group     xmrbuf_groups<>;
};

<CODE ENDS>

```

Buffers can be, and typically are, structured to contain multiple segments. Preposted receives that target a buffer uses a scatter list to place received messages in successive buffer segments.

An `xmrbs_seg` defines a single buffer segment. The fields included are:

- o `xmrseg_length` is the length of this contiguous buffer segment
- o `xmrseg_align` specifies the guaranteed alignment for the corresponding buffer segment.

- o `xmrseg_flags` which specify some noteworthy characteristics of the associated buffer segment.

The following flag bit is the only one currently defined:

- o `XMRSFLAG_PLT` indicates that the buffer segment in question is to be considered suitable as a target for data placement.

An `xmrbs_group` designates a set of buffer segment all with the same buffer segment characteristics as indicated by `xmr_grpinfo`. The buffer segments are contiguous within the buffer although they are likely not to be physically contiguous.

An `xmrbs_buf` defines a receiver's buffer structure and consists of multiple `xmrbs_groups`. This buffer structure, when made available as a transport property, allows the sender to structure transmissions so as to place DDP-eligible data in appropriate target buffer segments.

[7.3.](#) Message Data Placement Structures

These data structures show where in the virtual XDR stream for the set of messages, data is to be excised from that XDR stream and where that excised bulk data is to be found instead.

<CODE BEGINS>

```
union xmdp_loc switch(xmdp_type type)

    case XMPTYPE_EXRW:
        rpcrdma1_segment      xmdl_ex<>;
    case XMPTYPE_TBSN:
        xmdp_itemlen          xmdl_offset;
        xmdp_tbsn             xmdl_bsnum<>;
    case XMPTYPE_TOOSHORT:
    case XMPTYPE_NOITEM:
        void;
};

struct xmdp_mitem {
    xmdp_vsdisk      xmdmi_disp;
    xmdp_itemlen     xmdmi_length;
    xmdp_loc         xmdmi_where;
};

typedef xmdp_mitem      xmdp_grpinfo<>;

<CODE ENDS>
```

An `xmdp_loc` shows where a particular piece of bulk data is located. This information exists in multiple forms.

- o The case for DDP using explicit RDMA operations, contains, in `xmdl_ex` an array of `rpcrdma1_segments` showing where bulk data is

to be fetched from or has been transferred to.

- o The case for send-based data placement contains, in `xmdl_tbsn` an array placement-targetable buffer segments, indicating where bulk data, excised from the payload stream, is actually located. The bulk data starts `xmdl_offset` bytes into the buffer segment designated by `xmdl_bsnum[0]` and then proceeds through buffer segments denoted by successive `xmdl_bsnum` entries until the length of the data item is exhausted.
- o The cases for `XMPTYPE_TOO_SHORT` and `XMPTYPE_NOITEM` are only valid in responses

An `xmdp_mitem` denotes a specific item of bulk data. It consists of:

- o The XDR stream displacement of the bulk data excised from the payload stream, in `xmdmi_disp`.
- o The length of the data item, in `xmdmi_length`.
- o The actual location of the bulk data, in `xmdmi_loc`.

An `xmdp_grpinfo` consists of an array of `xmdp_mitems` describing all of the bulk data excised from all RPC messages sent in a single RPC-over-RDMA transmission group. Some possible cases:

- o The array is of length zero, indicating that there is no DDP-eligible data excised from the virtual XDR stream. In this case, the virtual XDR stream and the payload stream are identical.
- o The array consists of one or more `xmdp_mitems`, each of whose `xmdmi_where` fields is of type `XMPTYPE_EXRW`. In this case, the placement data corresponds to read chunks in the case in which a request is being sent and to write chunks in the case in which a reply is being sent.
- o The array consists of one or more `xmdp_mitems`, each of whose `xmdmi_where` fields is of type `XMPTYPE_TBSN`. In this case, each entry, whether it applies to bulk data in a request or a reply, describes data logically part of the message being sent, which may be part of any RPC-over-RDMA transmissions in the same transmission group.

- o The array consists of one or more xmdp_mitems, with xmdmi_where fields of a mixture of types, In this case, each entry, whether it applies to bulk data in a request or a reply, describes data logically part of the message being sent, although the method of getting access to that data may vary from entry to entry.

7.4. Response Direction Data Placement Structures

These data structures, when sent as part of the request, instruct the responder how to use data placement to place response data subject to special data placement.

<CODE BEGINS>

```
union xmdp_rsdloc switch(xmdp_type type)

    case XMPTYPE_EXRW:
    case XMPTYPE_CHOICE:
        rpcrdma1_segment      xmdrsdl_ex<>;
    case XMPTYPE_BYSIZE:
        xmdp_itemlen          xmdrsdl_dsdov;
        rpcrdma1_segment      xmdrsdl_bsex<>;
    case XMPTYPE_TBSN:
        void;
};

struct xmdp_rsdrange {
    xmdp_vsdisk      xmdrsdr_begin;
    xmdp_vsdisk      xmdrsdr_end;
};

struct xmdp_rsditem {
    xmdp_itemlen      xmdrsdi_minlen;
    xmdp_rsdloc       xmdrsdi_loc;
};

struct xmdp_rsdset {
    xmdp_rsdrange      xmdrsds_range;
    xmdp_rsditem        xmdrsds_items<>;
};

typedef xmdp_rsdset      xmdp_rsdgroup<>;
```

<CODE ENDS>

An xmdp_rsdloc contains information specifying where bulk data generated as part of a reply is to be placed. This information is defined as a union with the following cases:

Internet-Draft

RPC/RDMA Round-trip Reductions

December 2017

- o The case for DDP using explicit RDMA operations, XMPTYPE_EXRW, contains, in `xmrsdl_ex`, an array of `rpcrdma1_segments` showing where bulk data generated by the corresponding reply is to be transferred to.
- o The case allowing the responder to freely choose the data placement method, XMPTYPE_CHOICE, is identical. It also contains, in `xmrsdl_ex`, an array of `rpcrdma1_segments` showing where bulk data generated by the corresponding reply is to be transferred to if explicit RDMA requests are to be used.
- o The case for send-based data placement, XMPTYPE_TBSN, is void, since the decisions as to where bulk data is to be placed are made by the responder.
- o In the case directing the responder to choose the data placement method based on item size, XMPTYPE_BYSIZE, an array of `rpcrdma1_segments` is in `xmrsdl_bsex`.

In all cases, each `xmdp_rsdloc` sent as part of a request has a corresponding `xmdp_loc` in the associated response. The `xmdp_type` specified in the request will affect the type in the response, but the types are not necessarily the same. The table below describes the valid combinations of request and response `xmdp_type` values.

In this table, rows correspond to types in requests directing, the responder as to the desired placement in the response while the columns correspond to types in the ensuing response. Invalid combinations are labelled "Inv" while valid combination are labelled either "NDR" denoting no need to deregister memory, or "DR" to indicate that memory previously registered will need to be deregistered.

Type	EXRW	TBSN	TOOSHORT	NOITEM
EXRW	DR	Inv.	DR	DR
TBSN	Inv.	NDR	NDR	NDR
CHOICE	DR	NDR	DR	DR
BYSIZE	DR	NDR	DR	DR

Table 2

An `xmdp_rsdrange` denotes a range of positions in the XDR stream associated with a request. Particular directions regarding bulk data in the corresponding response are limited to such ranges, where

Noveck

Expires June 8, 2018

[Page 24]

Internet-Draft

RPC/RDMA Round-trip Reductions

December 2017

response XDR stream positions and request XDR stream positions can be reliably tied together.

When the ULP supports multiple individual operations per RPC request (e.g., COMPOUND and CB_COMPOUND in NFSv4), an `xmd_rsdrange` can isolate elements of the reply due to particular operations.

An `xmdp_rsditem` specifies the handling of one potential item of bulk data. The handling specified is qualified by a length range. If the item is smaller than `xmdrsdi_minlen`, it is not treated as bulk data and the corresponding data item appears in the payload stream, while that particular `xmdp_rsditem` is considered used up, making the next `xmdp_rsditem` in the `xmdp_rsdset` the target of the next DDP-eligible data item in the reply. Note that in the case in which `xmdrsdi_loc` specifies use of explicit RDMA operations, the area specified is not used and the requester is responsible for deregistering it.

For each `xmdp_rsditem`, there will be a corresponding `xmdp_mitem`

An `xmdp_rsdset` contains a set of `xmdp_rsditems` applicable to a given `xmdp_range` in the request.

An `xmdp_rsdgroup` designates a set of `xmdp_rsdsets` applicable to a particular RPC-over-RDMA transmission group. The `xmdrsds_range` fields of successive `xmdp_rsdsets` must be disjoint and in strictly increasing order.

[8.](#) Transport Properties

[8.1.](#) Property List

In this document we take advantage of the fact that the set of transport properties defined in [\[I-D.cel-nfsv4-rpcrdma-version-two\]](#). is subject to later extension. The additional transport properties are summarized below in Table 3.

In that table the columns have the following values:

- o The column labeled "property" identifies the transport property described by the current row.
- o The column labeled "#" specifies the propid value used to identify this property.
- o The column labeled "XDR type" gives XDR type of the data used to communicate the value of this property. This data overlays the nominally opaque field pv_data in a 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 "section" indicates the section (within this document) that explains the semantics and use of this transport property.

property	#	XDR type	default	section
RTR Support	3	uint32	0	8.2
Receive Buffer Structure	4	xmrbs_buf	Note1	8.3
Request Transmission Receive Limit	5	xms_grpxc	1	8.4
Response Transmission Send Limit	6	xms_grpxc	1	8.5

Table 3

The following notes apply to the above table:

1. The default value for the Receive Buffer Structure always consists of a single buffer segment, without any alignment restrictions and not targetable for DDP. The length of that buffer segment derives from the Receive Buffer Size Property if

available, and from the default receive buffer size otherwise.

[8.2.](#) RTR Support Property

<CODE BEGINS>

```
const uint32          XPROP_RTRSUPP = 3;
typedef uint32         xpr_rtrs;

const uint32          RTRS_XREQ = 1;
const uint32          RTRS_XRESP = 2;
const uint32          RTRS_XCONT = 4;
```

<CODE ENDS>

[8.3.](#) Receive Buffer Structure Property

This property defines the structure of the endpoint's receive buffers, in order to give a sender the ability to place bulk data in specific DDP-targetable buffer segments.

<CODE BEGINS>

```
const uint32          XPROP_RBSTRUCT = 4;
typedef xmrbs_buf     xpr_rbs;
```

<CODE ENDS>

Normally, this property, if specified, should be in agreement with Receive Buffer Size Property. However, the following rules apply.

- o If the value of Receive Buffer Structure Property is not specified, it is derived from the Receive Buffer Size Property, if known, and the default buffer size otherwise. The buffer is considered to consist of a single non-DDP-targetable segment whose size is the buffer size.
- o If the value of Receive Buffer Size Property is not specified and the Receive Buffer Structure Property is specified, the value of the former is derived from the latter, by adding up the length of all buffer segments specified.

[8.4.](#) Request Transmission Receive Limit Property

This property specifies the length of the longest request messages (in terms of number of transmissions) that a responder will accept.

<CODE BEGINS>

```
const uint32          XPROP_REQRXLIM = 5;
typedef uint32        xpr_rqrxl;
```

<CODE ENDS>

A requester can use this property to determine whether to send long requests by using message continuation or by using a position-zero read chunk.

[8.5.](#) Response Transmission Send Limit Property

This property specifies the length of the longest response message (in terms of number of transmissions) that a responder will generate.

<CODE BEGINS>

```
const uint32          XPROP_RESPSXLIM = 6;
typedef uint32        xpr_rssxl;
```

<CODE ENDS>

[9.](#) New Operations

[9.1.](#) Operations List

The proposed new operation are set for in Table 4 below. In that table, the columns have the following values:

- o The column labeled "operation" specifies the particular operation.
- o The column labeled "#" specifies the value of opttype for this operation.
- o The column labeled "XDR type" gives XDR type of the data structure used to describe the information in this new message type. This

data overlays the nominally opaque field `optinfo` in an `RDMA_OPTIONAL` message.

- o The column labeled "msg" indicates whether this operation is followed (or not) by an RPC message payload (or something else).
- o The column labeled "section" indicates the section (within this document) that explains the semantics and use of this optional operation.

operation	#	XDR type	msg	section
Transmit Request	5	<code>optxmt_req</code>	Note1	9.2
Transmit Response	6	<code>optxmt_resp</code>	Note1	9.3
Transmit Continue	7	<code>optxmt_cont</code>	Note2	9.4
Report Error	8	<code>optrept_err</code>	No.	9.5

Table 4

The following notes apply to the above table:

1. Contains an initial segment of the message payload stream for an RPC message, or the entire payload stream. The `optxr[qs]_pslen` field, indicates the length of the section present
2. May contain a part of a message payload stream for an RPC message, although not the entire payload stream. The `optxc_pslen` field, if non-zero, indicates that this portion is present, and the length of the section.

[9.2.](#) Transmit Request Operation

The message definition for this operation is as follows:

<CODE BEGINS>

```
const uint32      ROPT_XMTREQ = 1;
```

```

struct optxmt_req {
    xmdp_grpinfo      optxrq_dp;
    xmdp_rsdgroup     optxrq_rsd;
    xms_grpxc         optxrq_count;
    xms_grpxc         optxrq_rsbuf;
    xmdp_pldisp       optxrq_pslen;
};

```

<CODE ENDS>

The field `optxrq_dp` describes the fields in virtual XDR stream which have been excised in forming the payload stream, and information about where the corresponding bulk data is located.

The field `optxrq_rsd` consists of information directing the responder as to how to construct the reply, in terms of DDP. of length zero.

The field `optrq_count` specifies the count of transmissions in this group of transmissions used to send a request.

The field `optrq_repch` serves as a way to transfer a reply chunk to the responder to serve as a way in which a reply longer than the inline size limit may be transferred. Although, not prohibited by the protocol, it is unlikely to be used in environments in which message continuation is supported.

The field `optrq_pslen` gives the length of the payload stream for the RPC transmitted. The payload stream begins right after the end of the `optxmt_msg` and proceeds for `optxm_pslen` bytes. This can include crossing buffer segment boundaries.

[9.3.](#) Transmit Response Operation

The message definition for this operation is as follows:

<CODE BEGINS>

```
const uint32      ROPT_XMTRESP = 2;

struct optxmt_resp {
    xmdp_grpinfo    optxrs_dp;
    xms_grpxn       optxrs_count;
    xmdp_pldisp     optxrs_pslen;
};
```

<CODE ENDS>

The field `optxrs_dp` describes the fields in virtual XDR stream which have been excised in forming the payload stream, and information about where the corresponding bulk data is located.

The field `optxrs_count` specifies the count of transmissions in this group of transmissions used to send a reply.

The field `optxrs_pslen` gives the length of the payload stream for the RPC transmitted. The payload stream begins right after the end of the `optxmt_msg` and proceeds for `optxrs_pslen` bytes. This can include crossing buffer segment boundaries.

[9.4.](#) Transmit Continue Operation

RPC-over-RDMA headers of this type are used to continue RPC messages begun by RPC-over-RDMA message of type `ROPT_XMTREQ` or `ROPT_XMTRESP`. The `xid` field of this message must match that in the initial transmission.

This operation needs to be supported for the message continuation feature to be used.

The message definition for this operation is as follows:

<CODE BEGINS>

```
const uint32      ROPT_XMTCONT = 3;

struct optxmt_cont {
    xms_grpxn       optxc_xnum;
    uint32          optxc_itype;
    xmdp_pldisp     optxc_pslen;
};
```

<CODE ENDS>

The field `optxc_xnum` indicates the transmission number of this transmission within its transmission group.

The field `optxc_pslen` gives the length of the section of the payload stream which is located in the current RPC-over-RDMA transmission. It is valid for this length to be zero, indicating that there is no portion of the payload stream in this transmission. Except when the length is zero, the payload stream begins right after the end of the `optxmt_cont` and proceeds for `optxc_pslen` bytes. This can include crossing buffer segment boundaries. In any case, the payload streams for all transmissions within the same group are considered concatenated.

[9.5](#). Error Reporting Operation

This RPC-over-RDMA message type is used to signal the occurrence of errors that do not involve:

1. Transmission of a message that violates the rules specified in [\[I-D.cel-nfsv4-rpcrdma-version-two\]](#).
2. Transmission of a message described in this document which does not conform to the XDR specified here.
3. The transmission of a message, which, when assembled according to the rules here, cannot be decoded according to the XDR for the ULP.

Such errors can arise if the rules specified in this document are not followed and can be the result of a mismatch between multiple, each of which is valid when considered on its own.

The preliminary error-related definition is as follows:

<CODE BEGINS>

```
enum optr_err {
    OPTRERR_BADHMT = 1,
    OPTRERR_BADOMT = 2,
    OPTRERR_BADCONT = 3,
    OPTRERR_BADSEQ = 4,
    OPTRERR_BADXID = 5,
    OPTRERR_BADOFF = 6,
    OPTRERR_BADTBSN = 7,
    OPTRERR_BADPL = 8
}

union optr_info switch(optr_err optre_which) {

    case OPTRERR_BADHMT:
    case OPTRERR_BADOMT:
    case OPTRERR_BADSEQ:
    case OPTRERR_BADXID:
        uint32      optri_expect;
        uint32      optri_current;

    case OPTRERR_BADCONT:
        void;

    case OPTRERR_BADTBSN:
    case OPTRERR_BADOFF:
    case OPTRERR_BADPL:
        uint32      optri_value;
        uint32      optri_min;
        uint32      optri_max;

};

<CODE ENDS>
```

optr_err enumerates the various error conditions that might be reported.

- o OPTRERR_BADHMT indicates that a header message type other than the one expected was received. In this context, a particular message type can be considered "expected" only because of message or group continuation.
- o OPTRERR_BADOMT indicates that an optional message type other than the one expected was received. In this context, a particular

message type can be considered "expected" only because of message or group continuation.

- o OPTRERR_BADCONT indicates that a continuation messages was received when there was no reason to expect one.
- o OPTRERR_BADSEQ indicate that a transmission sequence number other than the one expected was received.
- o OPTRERR_BADXID indicate that an xid other than the one expected in a continuation context.
- o OPTRERR_BADTBSN indicate that an invalid target buffer sequence number was received.
- o OPTRERR_BADOFF indicate that a bad offset was received as part of an xmdp_loc. This is typically because the offset is larger than the buffer segment size.
- o OPTRERR_BADPL indicates that a bad offset was received for the payload length. This is typically because the length would make the area devoted to the payload stream not a subset of the actual transmission.

The optr_info gives error about the specific invalid field being reported. The additional information given depends on the specific error.

- o For the errors OPTRERR_BADHMT, OPTRERR_BADOMT, OPTRERR_BADSEQ, and

OPTRERR_BADXID, the expected and actual values of the field are reported

- o For the error OPTRERR_CONT, no additional information is provided.
- o For the errors OPTRERR_BADTBSN, OPTRERR_BADOFF, and OPTRERR_BADPL, the actual value together with a range of valid values is provided. When the actual value is within the valid range, it can be inferred that the actual value is not properly aligned (e.g. not on a 32-bit boundary)

The message definition for this operation is as follows:

<CODE BEGINS>

```
const uint32      ROPT_REPTERR = 4;

struct optrept_err {
    xms_id         optre_bad;
    xms_id         *optre_lead;
    optr_info      optre_info;
};
```

<CODE ENDS>

The field `optre_bad` is a description of the transmission on which the error was actually detected.

The optional field `optre_lead` is a description of an earlier transmission that might have led to the error reported.

The field `optre_info` provides information about the

This section contains an XDR [[RFC4506](#)] description of the proposed extension.

This description is provided in a way that makes it simple to extract into ready-to-use form. The reader can apply the following shell script to this document to produce a machine-readable XDR description of extension which can be combined with XDR for the base protocol to produce an XDR that includes the base protocol together with the 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 "ext.txt" then the reader can do the following to extract an XDR description file for this extension:

<CODE BEGINS>

```
sh extract.sh < ext.txt > xmitext.x
```

<CODE ENDS>

The XDR description for this extension can be combined with that for other extensions and that for the base protocol. While this is a complete description and can be processed by the XDR compiler, the result might not be usable to process the extended protocol, for a number of reasons:

The RPC-over-RDMA transport headers do not constitute an RPC program and version negotiation and message selection part of the

XDR, rather than being external to it.

Headers used for requests and replies are not necessarily paired, as they would be in an RPC program.

Header types defined as optional extensions overlay existing nominally opaque fields in the base protocol. While this overlay architecture allows code aware of the overlay relationships to have a more complete view of header structure, this overlay relationship cannot be expressed within the XDR language

[10.1](#). 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 author of the code is: 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>

<CODE BEGINS>

```
/// /*****
/// *****/
/// **
/// ** XDR for OPTIONAL protocol extension.
/// **
/// ** Includes support for both message continuation and send-based
/// ** DDP. The latter is supported by a new structure for the
/// ** specification of data placements which can be used for both
/// ** send-based data placement and DDP using explicit RDMA
/// ** operations.
/// **
/// ** Extensions include:
/// **
/// **     o Four new transport properties.
/// **     o Four new OPTIONAL message types
/// **
/// *****/
/// *****/
/// /*****
/// *
/// *           Core XDR Definitions
/// *
/// *****/

/// /*
/// * General XDR preliminaries for these features,
/// */
/// typedef uint32  xms_grpxn;
/// typedef uint32  xms_grpxc;
///
/// /*
/// * Basic XDR typedefs for the new approach to the specification of
/// * 8 data placement.
/// */
/// typedef uint32  xmdp_itemlen;
/// typedef uint32  xmdp_pldisp;
/// typedef uint32  xmdp_vsdisp;
/// typedef uint32  xmdp_tbsn;
///
/// /*
/// * Define the possible types of data placement items.
/// */
```

```
/// enum xmdp_type {
///     XMPTYPE_EXRW = 1,
///     XMPTYPE_TBSN = 2,
///     XMPTYPE_CHOOSE = 3,
///     XMPTYPE_BYSIZE = 4,
///     XMPTYPE_TOOSHORT = 5,
///     XMPTYPE_NOITEM = 6
/// };
///
/// /*
///  * XDR defining the placement of bulk items in the message being
///  * sent.
///  */
/// union xmdp_loc switch(xmdp_type type)
/// {
///     case XMPTYPE_EXRW:
///         rpcrdma1_segment      xmdl_ex<>;
///     case XMPTYPE_TBSN:
///         xmdp_itemlen          xmdl_offset;
///         xmdp_tbsn             xmdl_bsnum<>;
///     case XMPTYPE_TOOSHORT:
///     case XMPTYPE_NOITEM:
///         void;
/// };
///
///
///
/// struct xmdp_mitem {
///     xmdp_vsdisp      xmdmi_disp;
///     xmdp_itemlen     xmdmi_length;
///     xmdp_loc         xmdmi_where;
/// };
///
/// typedef xmdp_mitem      xmdp_grpinfo<>;
///
/// /*
///  * XDR defining the placement of bulk items in the response to the
///  * message being sent.
///  */
/// union xmdp_rsdloc switch(xmdp_type type)
/// {
///     case XMPTYPE_EXRW:
///     case XMPTYPE_CHOICE:
///         rpcrdma1_segment      xmdrsdl_ex<>;
///     case XMPTYPE_BYSIZE:
///         xmdl_itemlen          xmdrsdl_dsdo;
/// }
```

```

///          rpcrdma1_segment          xmdrsdl_bsex<>;
///          case XMPTYPE_TBSN:

```

Noveck

Expires June 8, 2018

[Page 38]

Internet-Draft

RPC/RDMA Round-trip Reductions

December 2017

```

///          void;
/// };
///
/// struct xmdp_rsdrange {
///          xmdp_vsdisp      xmdrsdr_begin;
///          xmdp_vsdisp      xmdrsdr_end;
/// };
///
/// struct xmdp_rsditem {
///          xmdp_itemlen      xmdrsdi_minlen;
///          xmdp_rsdloc       xmdrsdi_loc;
/// };
///
/// struct xmdp_rsdset {
///          xmdp_rsdrange     xmdrsds_range;
///          xmdp_rsditem      xmdrsds_items<>;
/// };
///
/// typedef xmdp_rsdset      xmdp_rsdgroup<>;
///
/// /*****
///  *
///  *          New Transport Properties
///  *
///  *****/
///
/// /*
///  * New Transport Property codes
///  */
/// const uint32          XPROP_RTRSUPP = 3;
/// const uint32          XPROP_RBSTRUCT = 4;
/// const uint32          XPROP_REQRXLIM = 5;
/// const uint32          XPROP_RESPSXLIM = 6;
///
/// /*
///  * XDR relating to RTR Support Property
///  */
/// typedef uint32          xpr_rtrs;
///

```

```

/// const uint32      RTRS_XREQ = 1;
/// const uint32      RTRS_XRESP = 2;
/// const uint32      RTRS_XCONT = 4;
///
/// /*
///  * Items related to Receive Buffer Structure Property
///  */
/// struct xmrbs_seg {
///      uint32          xmrseg_length;

```

Noveck

Expires June 8, 2018

[Page 39]

Internet-Draft

RPC/RDMA Round-trip Reductions

December 2017

```

///      uint32          xmrseg_align;
///      uint32          xmrseg_flags;
/// };
///
/// const uint32      XMRSFLAG_PLT = 0x01;
///
/// struct xmrbs_group {
///      uint32          xmrgrp_count;
///      xmrbs_seg       xmrgrp_info;
/// };
///
/// struct xmrbs_buf {
///      uint32          xmrbuf_length;
///      xmrbs_group     xmrbuf_groups<>;
/// };
/// typedef xmrbs_buf  xpr_rbs;
///
/// /*
///  * XDR relating to transmission limit properties
///  */
/// typedef uint32      xpr_rqrxl;
///
/// typedef uint32      xpr_rssxl;
///
/// /*****
///  *
///  *          New OPTIONAL Message Types
///  *
///  *****/
///
/// /*
///  * New message type codes

```

```

/// */
/// const uint32      ROPT_XMTREQ = 1;
/// const uint32      ROPT_XMTRESP = 2;
/// const uint32      ROPT_XMTCONT = 3;
/// const uint32      ROPT_REPTERR = 4;
///
///
/// /*
///  * New message type to do the initial transmission of a request.
///  */
/// struct optxmt_req {
///     xmdp_grpinfo    optxrq_dp;
///     xmdp_rsdgroup    optxrq_rsd;
///     xms_grpxc        optxrq_count;
///     xms_grpxc        optxrq_rsbuf;
///     xmdp_pldisp      optxrq_pslen;

```

Noveck

Expires June 8, 2018

[Page 40]

Internet-Draft

RPC/RDMA Round-trip Reductions

December 2017

```

///
/// };
///
/// /*
///  * New message type to do the initial transmission of a response.
///  */
/// struct optxmt_resp {
///     xmdp_grpinfo    optxrs_dp;
///     xms_grpxn        optxrs_count;
///     xmdp_pldisp      optxrs_pslen;
///
/// };
///
/// /*
///  * New message type to transmit the continuation of a request or
///  * response.
///  */
/// struct optxmt_cont {
///     xms_grpxn        optxc_xnum;
///     uint32            optxc_itype;
///     xmdp_pldisp      optxc_pslen;
/// };
///
/// /*
///  * XDR definitions to support error reporting.

```

```

/// */
/// enum optr_err {
///     OPTRERR_BADHMT = 1,
///     OPTRERR_BADOMT = 2,
///     OPTRERR_BADCONT = 3,
///     OPTRERR_BADSEQ = 4,
///     OPTRERR_BADXID = 5,
///     OPTRERR_BADOFF = 6,
///     OPTRERR_BADTBSN = 7,
///     OPTRERR_BADPL = 8
/// }
///
/// union optr_info switch(optr_err optre_which) {
///
///     case OPTRERR_BADHMT:
///     case OPTRERR_BADOMT:
///     case OPTRERR_BADSEQ:
///     case OPTRERR_BADXID:
///         uint32      optri_expect;
///         uint32      optri_current;
///
///     case OPTRERR_BADCONT:
///         void;

```

```

///
///
///     case OPTRERR_BADTBSN:
///     case OPTRERR_BADOFF:
///     case OPTRERR_BADPL:
///         uint32      optri_value;
///         uint32      optri_min;
///         uint32      optri_max;
///
/// };
///
/// struct xms_id {
///     uint32      xmsi_xid;
///     msg_type    xmsi_dir;
///     xms_grpxn   xmsi_seq;
/// };
///
/// /*

```

```

/// * New message type for error reporting.
/// */
/// struct optrept_err {
///     xms_id      optre_bad;
///     xms_id      *optre_lead;
///     optr_info    optre_info;
/// };
///
///
///
<CODE ENDS>

```

11. Security Considerations

The extension described has the same security considerations described in [\[RFC8166\]](#) and [\[I-D.cel-nfsv4-rpcrdma-version-two\]](#). With regard to the transport properties introduced in this document, it is possible that a man-in-the-middle could interfere with the communication of transport properties with possible negative effects. To prevent such interference, the steps described in [\[I-D.cel-nfsv4-rpcrdma-version-two\]](#) should be attended to.

The use of the techniques described in this document to reduce use of explicit RDMA operations raise important issues which implementers should consider:

While the use of these techniques may be expedient in certain cases, their use is not likely to be universal, at least for a considerable time. As a result, implementers should remain aware of the issues discussed in [Section 9.1 of \[RFC8166\]](#), unless and

until it is certain that none of a requesters memory can be registered for remote access.

Extra care needs to be taken in cases in which padding needs to be inserted in a transmission to ensure that DDP-targetable data item will be received in an appropriately aligned buffer segment. In some implementations, sensitive data could be inadvertently sent within the padding. To prevent this, the padding can be zeroed or it can be sent from a pre-zeroed area using a gather list.

12. IANA Considerations

This document does not require any 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>>.
- [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>>.

13.2. Informative References

- [I-D.cel-nfsv4-rpcrdma-version-two]
Lever, C. and D. Noveck, "RPC-over-RDMA Version 2 Protocol", [draft-cel-nfsv4-rpcrdma-version-two-05](#) (work in progress), July 2017.
- [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>>.

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

- [RFC5667] Talpey, T. and B. Callaghan, "Network File System (NFS) Direct Data Placement", [RFC 5667](#), DOI 10.17487/RFC5667, January 2010, <<https://www.rfc-editor.org/info/rfc5667>>.
- [RFC8178] Noveck, D., "Rules for NFSv4 Extensions and Minor Versions", [RFC 8178](#), DOI 10.17487/RFC8178, July 2017, <<https://www.rfc-editor.org/info/rfc8178>>.

Acknowledgments

The author gratefully acknowledges the work of Brent Callaghan and Tom Talpey producing the original RPC-over-RDMA Version One specification [[RFC5666](#)] and also Tom's work in helping to clarify that specification.

The author also wishes to thank Chuck Lever for his work resurrecting NFS support for RDMA in [[RFC8166](#)], for clarifying the relationship between RDMA and direct data placement, and for beginning the work on RPC-over-RDMA Version Two.

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

Author's Address

David Noveck
NetApp
1601 Trapelo Road
Waltham, MA 02451
US

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