Network File System Version 4
Internet-Draft
Intended status: Standards Track
Expires: December 1, 2017

C. Lever, Ed.
Oracle
D. Noveck
NetApp

May 30, 2017

# RPC-over-RDMA Version Two Protocol
# draft-cel-nfsv4-rpcrdma-version-two-04

## Abstract

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

## 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].

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 1, 2017.

## Copyright Notice

Table of Contents



## 1. Introduction

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

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

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

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

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

The major new feature in RPC-over-RDMA Version Two is extensibility of the RPC-over-RDMA header. Extensibility enables narrow changes to RPC-over-RDMA Version Two so that new optional capabilities can be introduced without a protocol version change and while maintaining interoperability with existing implementations.

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

As part of this new extensibility feature set, a mechanism for exchanging transport properties is introduced. This mechanism allows RPC-over-RDMA Version Two connection endpoints to communicate properties of their implementations, to request changes in properties of the other endpoint, and to notify peer endpoints of changes to properties that occur during operation.

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

Support for Remote Invalidation has been introduced into RPC-over-RDMA Version Two. An RPC-over-RDMA responder can now request invalidation of an STag as part of sending an RPC Reply, saving the

requester the effort of invalidating after message receipt. This new feature is general enough to enable a requester to control precisely when Remote Invalidation may be utilized by responders.

RPC-over-RDMA Version Two expands the repertoire of error codes to enable extensibility, support debugging, and to prevent requester retries when an error is permanent.

## 2. Inline Threshold

### 2.1. Terminology

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

A connection's inline threshold determines when RDMA Read or Write operations are required because the RPC message to be sent cannot be conveyed via RDMA Send and Receive. 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.

### 2.2. Motivation

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

Both registration and invalidation have a latency cost which 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 RDMA Read and Write operations are used for large payloads, and avoided for small payloads.

When RPC-over-RDMA Version One was conceived, the typical size of RPC messages that did not involve a significant data payload was under

500 bytes. A 1024-byte inline threshold adequately minimized the frequency of inefficient Long Calls and Replies.

Starting with NFSv4.1 [RFC5661], NFS COMPOUND RPC messages are larger and more complex than before. With a 1024-byte inline threshold, RDMA Read or Write operations are needed for frequent operations that do not bear a data payload, such as GETATTR and LOOKUP, reducing the efficiency of the transport.

To reduce the need to use Long Calls and Replies, RPC-over-RDMA Version Two increases the default inline threshold size. This also increases the maximum size of backward direction RPC messages.

## 2.3. Default Values

RPC-over-RDMA Version Two receiver implementations MUST support an inline threshold of 4096 bytes, but MAY support larger inline threshold values. A mechanism for discovering a peer's preferred inline threshold value (not defined in this document) may be used to optimize RDMA Send operations further. In the absense of such a mechanism, senders MUST assume a receiver's inline threshold is 4096 bytes.

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

# 3. Remote Invalidation

An STag that is registered using the FRWR mechanism (in a privileged execution context), or is registered via a Memory Window (in user space), may be invalidated remotely [RFC5040]. These mechanisms are available only 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 (ie. 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 Two 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).

### 3.1. Backward-Direction Remote Invalidation

Existing RPC-over-RDMA protocol specifications [I-D.ietf-nfsv4-rfc5666bis] [I-D.ietf-nfsv4-rpcrdma-bidirection] do not forbid direct data placement in the backward-direction, even though there is currently no Upper Layer Protocol that may use it.

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

However, in the backward 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.

## 4. Protocol Extensibility

The core RPC-over-RDMA Version Two header format is specified in Section 6 as a complete and stand-alone piece of XDR. Any change to this XDR description requires a protocol version number change.

### 4.1. Optional Features

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

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

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

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

## 4.2. Message Direction

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

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

To enable RDMA2_OPTIONAL type messages that do not carry an RPC message payload to be interpreted unambiguously, the rdma2_optional structure contains a field that identifies the message direction. A similar field has been added to the rpcrdma2_chunk_lists and rpcrdma2_error structures to simplify parsing the RPC-over-RDMA header at the receiver.

## 4.3. Documentation Requirements

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

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

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

- An explanation of the purpose and use of each new protocol element added
- An XDR description of the protocol elements, and a script to extract it
- A mechanism for reporting errors when the error is outside the available choices already available in the base protocol or in other extensions
- An indication of whether a Payload stream must be present, and a description of its contents
- A description of interactions with existing extensions

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

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

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

# 5. Transport Properties

## 5.1. Introduction To Transport Properties

### 5.1.1. Property 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:

- A code identifying the particular transport property being specified.

- A nominally opaque array which contains within it the XDR encoding of the specific property indicated by the associated code.

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 allow easier XDR extension of the set of properties by concatenating XDR 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.

A 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. The receiver of a message containing an rpcrdma2_propval MUST report an XDR error [ cel: which error? BAD_XDR, or do we want to add a new one? ] if the length of rdma_data is such that it extends beyond the bounds of the message transferred.

In cases in which the rpcrdma2_propid specified by rdma_which is understood by the receiver, the receiver also MUST report an XDR error if either of the following occur: [ cel: which error? BAD_XDR, or do we want to add a new one? ]

- The nominally opaque data within rdma_data is not valid when interpreted using the property-associated typedef.

- 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_propsubset 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.

Propvalsubsets are useful in a number of contexts:

- In the specification of transport properties at connection, they allow the sender to specify what subset of those are subject to later change.

- In responding to a request to modify a set of transport properties, they allow the responding endpoint to specify the subsets of those properties for which the requested change has been performed or been rejected.

### 5.1.2. Transport Property Groups

Transport properties are divided into a number of groups

- A basic set of transport properties defined in this document. See Section 5.2 for the complete list.

- Additional transport properties defined in future standards track documents as specified in Section 5.4.1.

- Experimental transport properties being explored preparatory to being considered for standards track definition. See the description in Section 5.4.2.

### 5.1.3. Operations Related to Transport Properties

There are a number of operations defined in Section 5.3 which are used to communicate and manage transport properties.

Prime among these is RDMA2_CONNPROP (defined in Section 5.3.1 which serves as a means by which an endpoint's transport properties may be presented to its peer, typically upon establishing a connection.

In addition, there are a set of related operations concerned with requesting, effecting and reporting changes in transport properties:

- RDMA2_REQPROP (defined in Section 5.3.2 which serves as a way for an endpoint to request that a peer change the values for a set of transport properties.

- RDMA2_RESPROP (defined in Section 5.3.3 is used to report on the disposition of each of the individual transport property changes requested in a previous RDMA2_REQPROP.

- RDMA2_UPDPROP (defined in Section 5.3.4 is used to report an unsolicited change in a transport property.

Unlike many other operation types, the above are not used to effect transfer of RPC requests but are internal one-way information transfers. However, a RDMA2_REQPROP and the corresponding RDMA2_RESPROP do constitute an RPC-like remote call. The other operations are not part of a remote call transaction.

## 5.2. Basic Transport Properties

Although the set of transport properties is subject to later extension, a basic set of transport properties is defined below in Table 1.

In that table, the columns contain the following information:

- The column labeled "property" identifies the transport property described by the current row.

- The column labeled "code" specifies the rpcrdma2_propid value used to identify this property.

- 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.

- 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.

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

| property | code | XDR type | default | section |
|---|---|---|---|---|
| Receive Buffer Size | 1 | uint32 | 4096 | 5.2.1 |
| Backward Request Support | 2 | enum rpcrdma2_bkreqsup | RDMA2_BKREQSUP_INLINE | 5.2.2 |

Table 1

Note that this table does not provide any indication regarding whether a particular property can change or whether a change in the value may be requested (see Section 5.3.2). Such matters are not addressed by the protocol definition. An implementation may provide information about its readiness to make changed in a particular property using the rdma_nochg field in the RDMA2_CONNPROP message.

A partner implementation can always request a change but peers MAY reject a request to change a property for any reason. Implementations are always free to reject such requests if they cannot or do not wish to effect the requested change.

Either of the following will result in effective rejection requests to change specific properties:

- If an endpoint does not wish to accept request to change particular properties, it may reject such requests as described in Section 5.3.3.

- If an endpoint does not support the RDMA2_REQPROP operation, the effect would be the same as if every request to change a set of property were rejected.

With regard to unrequested changes in transport properties, it is the responsibility of the implementation making the change to do so in a fashion that which does not interfere with the other partner's continued correct operation (see Section 5.2.1).

### 5.2.1. Receive Buffer Size

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

- Requesters may use the value to determine when it is necessary to provide a Position-Zero read chunk when sending a request.
- 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.
- 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.

Because there may be pre-posted receives with buffer sizes that reflect earlier values of the buffer size property, changing this property poses special difficulties:

- When the size is being raised, the partner should not be informed of the change until all pending receives using the older value have been eliminated.
- The size should not be reduced until the partner is aware of the need to reduce the size of future sends to conform to this reduced value. To ensure this, such a change should only occur in

response to an explicit request by the other endpoint (See Section 5.3.2). The participant making the request should use that lower size as the send size limit until the request is rejected (See Section 5.3.3) or an update to a size larger than the requested value becomes effective and the requested change is no longer pending (See Section 5.3.4).

### 5.2.2. Backward 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 backward-direction RPC requests.

<CODE BEGINS>

```
enum rpcrdma2_bkreqsup {
        RDMA2_BKREQSUP_NONE    = 0,
        RDMA2_BKREQSUP_INLINE  = 1,
        RDMA2_BKREQSUP_GENL    = 2
};

const uint32 RDMA2_PROPID_BRS = 2;
typedef rpcrdma2_bkreqsup rpcrdma2_prop_brs;
```

<CODE ENDS>

Multiple levels of support are distinguished:

- The value RDMA2_BKREQSUP_NONE indicates that receipt of backward-direction requests and replies is not supported.

- The value RDMA2_BKREQSUP_INLINE indicates that receipt of backward-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 backward direction requests or responses is not supported.

- The value RDMA2_BKREQSUP_GENL that receipt of backward-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 backward- 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 backward-direction replies is present on the server.

### 5.3. New Operations

The proposed new operations are set forth in Table 2 below. In that table, the columns contain the following information:

- The column labeled "operation" specifies the particular operation.
- The column labeled "code" specifies the value of opttype for this operation.
- 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 overlays the data portion of the nominally opaque field optinfo in an RDMA_OPTIONAL message.
- The column labeled "msg" indicates whether this operation is followed (or not) by an RPC message payload.
- The column labeled "section" indicates the section (within this document) that explains the semantics and use of this optional operation.

| operation | code | XDR type | msg | section |
|---|---|---|---|---|
| Specify Properties at Connection | 1 | optinfo_connprop | No | 5.3.1 |
| Request Property Modification | 2 | rpcrdma2_reqprop | No | 5.3.2 |
| Respond to Modification Request | 3 | rpcrdma2_resprop | No | 5.3.3 |
| Report Updated Properties | 4 | rpcrdma2_updprop | No | 5.3.4 |

Table 2

Support for all of the operations above is OPTIONAL. RPC-over-RDMA Version Two implementations that receive an operation that is not supported MUST respond with RDMA_ERROR message with an error code of RDMA_ERR_INVAL_OPTION.

The only operation support requirements are as follows:

- Implementations which send RDMA2_REQPROP messages must support RDMA2_RESPROP messages.

- Implementations which support RDMA2_RESPROP or RDMA2_UPDPROP messages must also support RDMA2_CONNPROP messages.

### 5.3.1. RDMA2_CONNPROP: Specify Properties at Connection

The RDMA2_CONNPROP message type allows an RPC-over-RDMA participant, 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_start;
        rpcrdma2_propsubset rdma_nochg;
};
```

<CODE ENDS>

All relevant transport properties that the sender is aware of should be included in rdma_start. Since support of this request is OPTIONAL, and since each of the properties is OPTIONAL as well, the sender cannot assume that the receiver will necessarily take note of these properties and so the sender should be prepared for cases in which the partner continues to assume that the default value for a particular property is still in effect.

Values of the subset of transport properties specified by rdma_nochg is not expected to change during the lifetime of the connection.

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 partners 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_start 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.

### 5.3.2. RDMA2_REQPROP: Request Modification of Properties

The RDMA2_REQPROP message type allows an RPC-over-RDMA participant, whether client or server, to request of its partner that relevant transport properties be changed.

The rdma_xid field allows the request to be tied to a corresponding response of type RDMA2_RESPROP (See Section 5.3.3.) In assigning the value of this field, the sender does not need to avoid conflict with xid's associated with RPC messages or with RDMA2_REQPROP messages sent by the peer endpoint.

The partner need not change the properties as requested by the sender but if it does support the message type, it will generate a RDMA2_RESPROP message, indicating the disposition of the request.

The message definition for this operation is as follows:

<CODE BEGINS>

```
struct rpcrdma2_reqprop {
       rpcrdma2_propset rdma_want;
};
```

<CODE ENDS>

The rpcrdma2_propset rdma_want is a set of transport properties together with the desired values requested by the sender.

### 5.3.3. RDMA2_RESPROP: Respond to Request to Modify Transport Properties

The RDMA2_RESPROP message type allows an RPC-over-RDMA participant to respond to a request to change properties by its partner, indicating how the request was dealt with.

The message definition for this operation is as follows:

```
<CODE BEGINS>

struct rpcrdma2_resprop {
        rpcrdma2_propsubset rdma_done;
        rpcrdma2_propsubset rdma_rejected;
        rpcrdma2_propset rdma_other;
};

<CODE ENDS>
```

The rdma_xid field of this message must match that used in the RDMA2_REQPROP message to which this message is responding.

The rdma_done field indicates which of the requested transport property changes have been effected as requested. For each such property, the receiver is entitled to conclude that the requested change has been made and that future transmissions may be made based on the new value.

The rdma_rejected field indicates which of the requested transport property changes have been rejected by the sender. This may be because of any of the following reasons:

- The particular property specified is not known or supported by the receiver of the RDMA2_REQPROP message.
- The implementation receiving the RDMA2_REQPROP message does not support modification of this property.
- The implementation receiving the RDMA2_REQPROP message has chosen to reject the modification for another reason.

The rdma_other field contains new values for properties where a change is requested. The new value of the property is included and may be a value different from the original value in effect when the change was requested and from the requested value. This is useful when the new value of some property is not as large as requested but still different from the original value, indicating a partial satisfaction of the peer's property change request.

The sender MUST NOT include rpcrdma2_propval items within rdma_other that are for properties other than the ones for which the corresponding property request has requested a change. If the receiver finds such a situation, it MUST ignore the erroneous rpcrdma2_propval items.

The subsets of properties specified by rdma_done, rdma_rejected, and included in rdma_other MUST NOT overlap, and when ored together, should cover the entire set of properties specified by rdma_want in the corresponding request. If the receiver finds such an overlap or mismatch, it SHOULD treat properties missing or within the overlap as having been rejected.

### 5.3.4. RDMA2_UPDPROP: Update Transport Properties

The RDMA2_UPDPROP message type allows an RPC-over-RDMA participant to notify the other participant that a change to the transport properties has occurred. This is because the sender has decided, independently, to modify one or more transport properties and is notifying the receiver of these changes.

The message definition for this operation is as follows:

<CODE BEGINS>

```
struct rpcrdma2_updprop {
        rpcrdma2_propset rdma_now;
};
```

<CODE ENDS>

rdma_now defines the new property values to be used.

## 5.4. Extensibility

### 5.4.1. Additional Properties

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.

- The rpcrdma2_propid value used to identify this property.
- The XDR typedef specifying the form in which the property value is communicated.

- A description of the transport property that is communicated by the sender of RDMA2_CONNPROP and RDMA2_UPDPROP and requested by the sender of RDMA2_REQPROP.

- An explanation of how this knowledge could be used by the participant receiving this information.

- Information giving rules governing possible changes of values of this property.

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 working group should insure that:

- 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.

- rpcrdma2_propid values specified in the document do not conflict with the range reserved for experimental use, as defined in Section 5.4.2.

Documents defining new properties fall into a number of categories.

- Those defining new properties and explaining (only) how they affect use of existing message types.

- Those defining new OPTIONAL message types and new properties applicable to the operation of those new message types.

- 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.

### 5.4.2. Experimental Properties

Given the design of the transport properties data structure, it possible to use the operations to implement experimental, possibly unpublished, transport properties.

rpcrdma2_propid values in the range from 4,294,967,040 to 4,294,967,295 are reserved for experimental use and these values should not be assigned to new properties in standards track documents.

When values in this range are used there is no guarantee if successful interoperation among independent implementations.

## 6. XDR Protocol Definition

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

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

```
<CODE BEGINS>

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

<CODE ENDS>
```

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

```
<CODE BEGINS>

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

<CODE ENDS>
```

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

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

<CODE ENDS>

### 6.2. RPC-Over-RDMA Version Two XDR

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

<CODE BEGINS>

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

```
   ///
   /// enum rpcrdma2_errcode {
   ///         RDMA2_ERR_VERS = 1,
   ///         RDMA2_ERR_BAD_XDR = 2,
   ///         RDMA2_ERR_INVAL_PROC = 3,
   ///         RDMA2_ERR_READ_CHUNKS = 4,
   ///         RDMA2_ERR_WRITE_CHUNKS = 5,
   ///         RDMA2_ERR_SEGMENTS = 6,
   ///         RDMA2_ERR_WRITE_RESOURCE = 7,
   ///         RDMA2_ERR_REPLY_RESOURCE = 8,
   ///         RDMA2_ERR_INVAL_OPTION = 9,
   ///         RDMA2_ERR_SYSTEM = 10,
   /// };
   ///
   /// 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_BAD_XDR:
   ///           void;
   ///         case RDMA2_ERR_INVAL_PROC:
   ///           void;
   ///         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;
   ///         case RDMA2_ERR_INVAL_OPTION:
   ///           void;
   ///         case RDMA2_ERR_SYSTEM:
   ///           void;
   /// };
   ///
   /// struct rpcrdma2_optional {
```

```
   ///         enum msg_type rdma_optdir;
   ///         uint32 rdma_opttype;
   ///         opaque rdma_optinfo<>;
   /// };
   ///
   /// typedef rpcrdma2_propid uint32;
   ///
   /// struct rpcrdma2_propval {
   ///         rpcrdma2_propid rdma_which;
   ///         opaque          rdma_data<>;
   /// };
   ///
   /// typedef rpcrdma2_propval rpcrdma2_propset<>;
   /// typedef uint32 rpcrdma2_propsubset<>;
   ///
   /// struct rpcrdma2_connprop {
   ///         rpcrdma2_propset rdma_start;
   ///         rpcrdma2_propsubset rdma_nochg;
   /// };
   ///
   /// struct rpcrdma2_reqprop {
   ///        rpcrdma2_propset rdma_want;
   /// };
   ///
   /// struct rpcrdma2_resprop {
   ///        rpcrdma2_propsubset rdma_done;
   ///        rpcrdma2_propsubset rdma_rejected;
   ///        rpcrdma2_propset rdma_other;
   /// };
   ///
   /// struct rpcrdma2_updprop {
   ///        rpcrdma2_propset rdma_now;
   /// };

   /// enum rpcrdma2_proc {
   ///         RDMA2_MSG = 0,
   ///         RDMA2_NOMSG = 1,
   ///         RDMA2_ERROR = 4,
   ///         RDMA2_OPTIONAL = 5,
   ///         RDMA2_CONNPROP = 6,
   ///         RDMA2_REQPROP = 7,
   ///         RDMA2_RESPROP = 8,
   ///         RDMA2_UPDPROP = 9
   /// };
   ///
   /// union rpcrdma2_body switch (rpcrdma2_proc rdma_proc) {
   ///         case RDMA2_MSG:
   ///           rpcrdma2_chunk_lists rdma_chunks;
```

```
   ///         case RDMA2_NOMSG:
   ///           rpcrdma2_chunk_lists rdma_chunks;
   ///         case RDMA2_ERROR:
   ///           rpcrdma2_error rdma_error;
   ///         case RDMA2_OPTIONAL:
   ///           rpcrdma2_optional rdma_optional;
   ///         case RDMA2_CONNPROP:
   ///           rpcrdma2_connprop rdma_connprop;
   ///         case RDMA2_REQPROP:
   ///           rpcrdma2_reqprop rdma_reqprop;
   ///         case RDMA2_RESPROP:
   ///           rpcrdma2_resprop rdma_resprop;
   ///         case RDMA2_UPDPROP:
   ///           rpcrdma2_updprop rdma_updprop;
   /// };
   ///
   /// struct rpcrdma2_xprt_hdr {
   ///           uint32 rdma_xid;
   ///           uint32 rdma_vers;
   ///           uint32 rdma_credit;
   ///           rpcrdma2_body rdma_body;
   /// };
   ///
   /// /*
   ///  * Transport propid values for basic properties
   ///  */
   /// const uint32 RDMA2_PROPID_RBSIZ = 1;
   /// const uint32 RDMA2_PROPID_BRS = 2;
   ///
   /// /*
   ///  * Transport property typedefs
   ///  */
   /// typedef uint32 rpcrdma2_prop_rbsiz;
   /// typedef rpcrdma2_bkreqsup rpcrdma2_prop_brs;
   ///
   /// enum rpcrdma2_bkreqsup {
   ///         RDMA2_BKREQSUP_NONE = 0,
   ///         RDMA2_BKREQSUP_INLINE = 1,
   ///         RDMA2_BKREQSUP_GENL = 2
   /// };
```

<CODE ENDS>

### 6.2.1. Presence Of Payload

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

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

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

### 6.2.2. Message Direction

Implementations of RPC-over-RDMA Version Two are REQUIRED to support backwards direction operation as described in [I-D.ietf-nfsv4-rpcrdma-bidirection]. RPC-over-RDMA Version Two introduces the rdma_direction field in its transport header to optimize the process of distinguishing between forward- and backwards-direction messages.

The rdma_direction field qualifies the value contained in the transport header's rdma_xid field. This enables a receiver to reliably avoid performing an XID lookup on incoming backwards-direction Call messages.

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 rdma_direction field to REPLY (1). Otherwise the rdma_direction field is set to CALL (0). For example:

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

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

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

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

### 6.2.3. Remote Invalidation

To request Remote Invalidation, a requester MUST set 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 Call may be invalidated by the responder, the requester MUST set 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 MUST use 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 MUST use RDMA Send With Invalidate to transmit that RPC reply, and MUST use the value in the RPC Call's rdma_inv_handle field to construct the Send With Invalidate Work Request.

### 6.2.4. Transport Errors

Error handling works the same way in RPC-over-RDMA Version Two as it does in RPC-over-RDMA Version One, with the addition of several new error codes, and error messages never flow from requester to responder. Version One error handling is described in Section 5 of [I-D.ietf-nfsv4-rfc5666bis].

In all cases below, the responder copies the values of the rdma_xid and rdma_vers fields from the incoming transport header that generated the error to transport header of the error response. The responder sets the rdma_proc field to RDMA2_ERROR, and the rdma_credit field is set to the credit grant value for this connection.

RDMA2_ERR_VERS
: This is the equivalent of ERR_VERS in RPC-over-RDMA Version One. The error code value, semantics, and utilization are the same.

RDMA2_ERR_INVAL_PROC
: If a responder recognizes the value in the rdma_vers field, but it does not recognize the value in the rdma_proc field, it MUST set the rdma_err field to RDMA2_ERR_INVAL_PROC.

RDMA2_ERR_BAD_XDR

If a responder recognizes the values in the rdma_vers and 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. 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 One. 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.

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.

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 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_INVAL_OPTION

A responder MUST set the rdma_err field to RDMA2_ERR_INVAL_OPTION when an RDMA2_OPTIONAL message is received and the responder does not recognize the value in the rdma_opttype field.

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_xid field.

## 7. Protocol Version Negotiation

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

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

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 One, this initial message can be no larger than the Version One default inline threshold of 1024 bytes.

### 7.1. Server Does Support RPC-over-RDMA Version Two

If the server does support RPC-over-RDMA Version Two, 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 Two connections (4096 bytes).

### 7.2. Server Does Not Support RPC-over-RDMA Version Two

If the server does not support RPC-over-RDMA Version Two, it MUST send an RPC-over-RDMA message to the client with the same XID, with RDMA2_ERROR in the rdma_proc field, and with the error code RDMA2_ERR_VERS. This message also reports a range of protocol versions that the 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 Two. A client can assume the same situation (no server support for RPC-over-RDMA Version Two) 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 Two

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.

### 7.4. Security Considerations

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

#### 7.4.1. 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.

## 8. IANA Considerations

This document does not require actions by IANA.

## 9. References

### 9.1. Normative References

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

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

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

### 9.2. Informative References

[I-D.ietf-nfsv4-rfc5666bis] Lever, C., Simpson, W., and T. Talpey, "Remote Direct Memory Access Transport for Remote Procedure Call, Version One", draft-ietf-nfsv4-rfc5666bis-11 (work in progress), March 2017.

[I-D.ietf-nfsv4-rpcrdma-bidirection] Lever, C., "Bi-directional Remote Procedure Call On RPC-over-RDMA Transports", draft-ietf-nfsv4-rpcrdma-bidirection-08 (work in progress), March 2017.

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

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

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

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

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

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

## Appendix A. Acknowledgments

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

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

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

## Authors' Addresses

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

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

David Noveck
NetApp
1601 Trapelo Road
Waltham, MA 02451
USA

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