

Network File System Version 4  
Internet-Draft  
Intended status: Standards Track  
Expires: March 28, 2017

D. Noveck  
HPE  
September 24, 2016

RPC-over-RDMA Extension to Manage Transport Properties  
draft-dnoveck-nfsv4-rpcrdma-xcharext-03

## Abstract

This document specifies an extension RPC-over-RDMA Version Two. The extension enables endpoints of an RPC-over-RDMA connection to exchange information which can be used to optimize message transfer.

## 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 March 28, 2017.

## Copyright Notice

Copyright (c) 2016 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 (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

<a href="#">1.</a>	<a href="#">Preliminaries</a>	<a href="#">2</a>
<a href="#">1.1.</a>	<a href="#">Requirements Language</a>	<a href="#">2</a>
<a href="#">1.2.</a>	<a href="#">Introduction</a>	<a href="#">2</a>
<a href="#">1.3.</a>	<a href="#">Role Terminology</a>	<a href="#">3</a>
<a href="#">2.</a>	<a href="#">Transport Properties</a>	<a href="#">4</a>
<a href="#">2.1.</a>	<a href="#">Property Model</a>	<a href="#">4</a>
<a href="#">2.2.</a>	<a href="#">Transport Property Groups</a>	<a href="#">5</a>
<a href="#">2.3.</a>	<a href="#">Operations Related to Transport Properties</a>	<a href="#">6</a>
<a href="#">3.</a>	<a href="#">Basic Transport Properties</a>	<a href="#">6</a>
<a href="#">3.1.</a>	<a href="#">Receive Buffer Size</a>	<a href="#">8</a>
<a href="#">3.2.</a>	<a href="#">Requester Remote Invalidation</a>	<a href="#">9</a>
<a href="#">3.3.</a>	<a href="#">Backward Request Support</a>	<a href="#">10</a>
<a href="#">4.</a>	<a href="#">New Operations</a>	<a href="#">11</a>
<a href="#">4.1.</a>	<a href="#">ROPT_CONNPROP: Specify Properties at Connection</a>	<a href="#">12</a>
<a href="#">4.2.</a>	<a href="#">ROPT_REQPROP: Request Modification of Properties</a>	<a href="#">13</a>
<a href="#">4.3.</a>	<a href="#">ROPT_RESPROP: Respond to Request to Modify Transport Properties</a>	<a href="#">13</a>
<a href="#">4.4.</a>	<a href="#">ROPT_UPDPROP: Update Transport Properties</a>	<a href="#">15</a>
<a href="#">5.</a>	<a href="#">XDR</a>	<a href="#">15</a>
<a href="#">5.1.</a>	<a href="#">Code Component License</a>	<a href="#">16</a>
<a href="#">5.2.</a>	<a href="#">XDR Proper for Extension</a>	<a href="#">18</a>
<a href="#">6.</a>	<a href="#">Extensibility</a>	<a href="#">19</a>
<a href="#">6.1.</a>	<a href="#">Additional Properties</a>	<a href="#">19</a>
<a href="#">6.2.</a>	<a href="#">Experimental Properties</a>	<a href="#">20</a>
<a href="#">7.</a>	<a href="#">Security Considerations</a>	<a href="#">21</a>
<a href="#">8.</a>	<a href="#">IANA Considerations</a>	<a href="#">21</a>
<a href="#">9.</a>	<a href="#">References</a>	<a href="#">21</a>
<a href="#">9.1.</a>	<a href="#">Normative References</a>	<a href="#">21</a>
<a href="#">9.2.</a>	<a href="#">Informative References</a>	<a href="#">22</a>
<a href="#">Appendix A.</a>	<a href="#">Acknowledgments</a>	<a href="#">22</a>
<a href="#">Author's Address</a>		<a href="#">22</a>

[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 specifies an extension to RPC-over-RDMA Version Two. It allows each participating endpoint on a single connection to communicate various properties of its implementation, to request

changes in properties of the other endpoint, and to notify the other endpoint of changes to properties made during operation.

The extension described herein specifies OPTIONAL message header types to implement this mechanism. The means by which the implementation support status of these OPTIONAL types is ascertained is described in [rpccrmdav2].

Although this document specifies the new OPTIONAL message header types to implement these functions, the precise means by which the presence of support for these OPTIONAL functions will be ascertained is not described here, as would be done more appropriately by the RFC defining a version of RPC-over-RDMA which supports protocol extension.

This document is currently written to conform to the extension model for RPC-over-RDMA Version Two as described in [[rpccrdmav2](#)].

### [1.3.](#) Role 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 are fixed 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.

The roles of sender and receiver change from message to message. With regard to the types of messages described in this document, both the client and the server can act as sender and receiver. With regard to messages used to transfer RPC requests and replies, the requester sends requests and receives replies while the responder receives requests and sends replies.

## [2.](#) Transport Properties

### [2.1.](#) 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:

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

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

<CODE BEGINS>

```
typedef propid          uint32;

struct propval {
    propid              pv_which;
    opaque              pv_data<>;
};

typedef propval          propvalset<>;
```

```
typedef uint32          propvalsubset<>;
```

```
<CODE ENDS>
```

A 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.

A propval specifies a value of a particular transport property with the particular property identified by pv\_which, while the associated value of that property is contained within pv\_data.

A pv\_data which is of zero length is interpreted as indicating the default value or the property indicated by pv\_which.

While pv\_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 pv\_which. The receiver of a message containing a propval MUST report an XDR error if the length of pv\_data is such that it extends beyond the bounds of the message transferred.

In cases in which the propid specified by pv\_which is understood by the receiver, the receiver also MUST report an XDR error if either of the following occur:

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

Note that no error is to be reported if pv\_which is unknown to the receiver. In that case, that propval is not processed and processing continues using the next propval, if any.

A propvalset specifies a set of transport properties. No particular ordering of the propvals within it is imposed.

A `propvalsubset` identifies a subset of the properties in a previously specified `propvalset`. Each bit in the mask denotes a particular element in a previously specified `propvalset`. If a particular `propval` is at position `N` in the array, then bit number `N mod 32` in word `N div 32` specifies whether that particular `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:

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

## [2.2.](#) Transport Property Groups

Transport properties are divided into a number of groups

- o A basic set of transport properties defined in this document. See [Section 3](#) for the complete list.

- o Additional transport properties defined in future standards track documents as specified in [Section 6.1](#).
- o Experimental transport properties being explored preparatory to being considered for standards track definition. See the description in [Section 6.2](#).

## [2.3.](#) Operations Related to Transport Properties

There are a number of operations defined in [Section 4](#) which are used to communicate and manage transport properties.

Prime among these is `ROPT_CONNPROP` (defined in [Section 4.1](#) 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:

- o ROPT\_REQPROP (defined in [Section 4.2](#) which serves as a way for an endpoint to request that a peer change the values for a set of transport properties.
- o ROPT\_RESPROP (defined in [Section 4.3](#) is used to report on the disposition of each of the individual transport property changes requested in a previous ROPT\_REQPROP.
- o ROPT\_UPDPROP (defined in [Section 4.4](#) 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 ROPT\_REQPROP and the corresponding ROPT\_RESPROP do constitute an RPC-like remote call. The other operations are not part of a remote call transaction.

### [3.](#) 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:

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

- o The column labeled "code" specifies the propid value used to identify this property.
- o The column labeled "XDR type" gives the XDR type of the data used to communicate the value of this property. This data type overlays the data portion of the nominally opaque field 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	code	XDR type	default	section
Receive Buffer Size	1	uint32	4096	3.1
Requester Remote Invalidation	2	bool	false	3.2
Backward Request Support	3	enum bkreqsup	BKREQSUP_INLINE	3.3

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 4.2](#)). 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 `optconnpr_nochg` field in the `ROPT_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:

- o If an endpoint does not wish to accept request to change



particular properties, it may reject such requests as described in [Section 4.3](#).

- o If an endpoint does not support the ROPT\_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 3.1](#)).

### [3.1](#). 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    PROP_RBSIZ = 1;
typedef uint32  prop_rbsiz;
```

<CODE ENDS>

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

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

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:

- o 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.
- o 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 4.2](#)). The participant making the request should use that lower size as the send size limit until the request is rejected (See [Section 4.3](#)) or an update to a size larger than the requested value becomes effective and the requested change is no longer pending (See [Section 4.4](#)).

### [3.2](#). Requester Remote Invalidation

The Requester Remote Invalidation property indicates that the current endpoint, when in the role of a requester, is prepared for the responder to use RDMA Send With Invalidate when replying to an RPC-over-RDMA request containing non-empty chunk lists.

As RPC-over-RDMA is currently used, memory registrations exposed to peers are not established by the server and explicit RDMA operations are not done to satisfy backward direction requests. This makes it unlikely that servers will present non-default values of the PROP\_REQREMINV property or that clients will take note of that value when presented by servers.

<CODE BEGINS>

```
const uint32    PROP_REQREMINV = 2;
typedef bool    prop_reqreminv;
```

<CODE ENDS>

When the Requester Remote Invalidate property is set to false, a responder MUST use Send to convey RPC reply messages to the requester. When the Requester Remote Invalidate property is set to true, a responder MAY use Send With Invalidate instead of Send to convey RPC replies to the requester.

The value of the Requester Remote Invalidate property is not likely to change from the value reported by ROPT\_INITPROP (see [Section 4.2](#)).

### [3.3](#). 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 bkreqsup {
    BKREQSUP_NONE      = 0,
    BKREQSUP_INLINE    = 1,
    BKREQSUP_GENL      = 2
};
```

```
const uint32      PROP_BRS = 3;
typedef bkreqsup prop_brs;
```

<CODE ENDS>

Multiple levels of support are distinguished:

- o The value BKREQSUP\_NONE indicates that receipt of backward-direction requests and replies is not supported.
- o The value 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.
- o The value 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.

#### 4. New Operations

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

- o The column labeled "operation" specifies the particular operation.
- o The column labeled "code" specifies the value of opttype for this operation.
- o The column labeled "XDR type" gives the XDR type of the data structure used to describe the information in this new message type. This data overlays the data portion of 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.
- o 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	4.1
Request Property Modification	2	optinfo_reqprop	No	4.2
Respond to Modification Request	3	optinfo_resprop	No	4.3
Report Updated	4	optinfo_updprop	No	4.4

Properties				
+-----+	+-----+	+-----+	+-----+	+-----+

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\_INVALID\_OPTION as specified in [[rpcrdmav2](#)]

The only operation support requirements are as follows:

- o Implementations which send ROPT\_REQPROP messages must support ROPT\_RESPROP messages.

- o Implementations which support ROPT\_RESPROP or ROPT\_UPDPROP messages must also support ROPT\_CONNPROP messages.

#### [4.1](#). ROPT\_CONNPROP: Specify Properties at Connection

The ROPT\_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>

```
const uint32      ROPT_CONNPROP= 1;

struct optinfo_connprop {
    propvalset      opticonnpr_start;
    propvalsubset   opticonnpr_nochg;
};
```

<CODE ENDS>

All relevant transport properties that the sender is aware of should be included in opticonnpr\_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 `opticonnpr_nochg` is not expected to change during the lifetime of the connection.

Generally, a participant will send a `ROPT_CONNPR` 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. See [\[rpcrdmav2\]](#) for details.

Properties not included in `opticonnpr_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 `ROPT_CONNPR` 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.

#### [4.2.](#) `ROPT_REQPROP`: Request Modification of Properties

The `ROPT_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 `ROPT_RESPROP` (See [Section 4.3.](#)) In assigning the value of this field, the sender does not need to avoid conflict with `xid`'s associated with RPC messages or with `ROPT_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

ROPT\_RESPROP message, indicating the disposition of the request.

The message definition for this operation is as follows:

<CODE BEGINS>

```
const uint32      ROPT_REQPROP = 2;

struct optinfo_reqprop {
    propvalset      optreqpr_want;
};
```

<CODE ENDS>

The propvalset optreqpr\_want is a set of transport properties together with the desired values requested by the sender.

#### [4.3.](#) ROPT\_RESPROP: Respond to Request to Modify Transport Properties

The ROPT\_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>

```
const uint32      ROPT_RESPROP = 3;

struct optinfo_resprop {
    propvalsubset  optrespr_done;
    propvalsubset  optrespr_rej;
    propvalset     optrespr_other;
};
```

<CODE ENDS>

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

The `optrespr_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 `optrespr_rej` 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:

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

The `optrespr_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 propvals within `optrespr_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 propvals.

The subsets of properties specified by `optrespr_done`, `optrespr_rej`, and included in `optrespr_other` **MUST NOT** overlap, and when ored together, should cover the entire set of properties specified by `optreqpr_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.

#### [4.4.](#) ROPT\_UPDPROP: Update Transport Properties

The ROPT\_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>

```
const uint32      ROPT_UPDPROP = 4;

struct optinfo_updprop {
    propvalset      optupdpr_now;
};
```

<CODE ENDS>

optupdpr\_now defines the new property values to be used.

#### [5.](#) XDR

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 combines the base protocol 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 > charext.x
```

<CODE ENDS>

### [5.1.](#) 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.

Internet-Draft

RPC-over-RDMA Transport Properties

September 2016

&lt;CODE BEGINS&gt;

```
/// /*
///  * 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>
```

## [5.2.](#) XDR Proper for Extension

```
<CODE BEGINS>

///
////*
/// * Core transport property types
/// */
///typedef propid          uint32;
///
///struct propval {
///    propid          pv_which;
///    opaque          pv_data<>;
///};
///
///typedef propval          propvalset<>;
///
///typedef uint32           propvalsubset<>;
///
////*
/// * Transport property codes for basic properties
/// */
///const uint32    PROP_RBSIZ = 1;
///const uint32    PROP_REQREMINV = 2;
///const uint32    PROP_BRS = 3;
///
////*
/// * Other transport property types
/// */
///enum bkreqsup {
///    BKREQSUP_NONE    = 0,
///    BKREQSUP_INLINE  = 1,
```

```

///          BKREQSUP_GENL      = 2
///};
///
////*
/// * Transport property typedefs
/// */
///typedef uint32    prop_bsiz;
///typedef bool      prop_reqreminv;
///typedef bkreqsup  prop_brs;
///
////*
/// * Optional operation codes
/// */
///const uint32      ROPT_CONNPROP = 1;
///const uint32      ROPT_REQPROP = 2;

```

```

///const uint32      ROPT_RESPRO = 3;
///const uint32      ROPT_UPDPROP = 4;
///
////*
/// * Optional operation message structures
/// */
///struct optinfo_connprop {
///    propvalset      opticonnpr_start;
///    propvalsubset   opticonnpr_nochg;
///};
///
///struct optinfo_reqprop {
///    propvalset      optreqpr_want;
///};
///
///struct optinfo_resprop {
///    propvalsubset   optrespr_done;
///    propvalsubset   optrespr_rej;
///    propvalset      optrespr_other;
///};
///
///struct optinfo_updprop {
///    propvalset      optupdpr_now;
///};

```

<CODE ENDS>

## [6.](#) Extensibility

### [6.1.](#) 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.

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

- o A description of the transport property that is communicated by the sender of ROPT\_INITXCH and ROPT\_UPDXCH and requested by the sender of ROP\_REQXCH.
- o An explanation of how this knowledge could be used by the participant receiving this information.
- o 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:

- o The propids specified in the document do not conflict with those

currently assigned or in use by other pending working group documents defining transport properties.

- o The propids specified in the document do not conflict with the range reserved for experimental use, as defined in [Section 6.2](#).

Documents defining new properties fall into a number of categories.

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

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

## [6.2](#). Experimental Properties

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

propids 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.

## [7](#). Security Considerations

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 any actions by IANA.

## 9. References

### 9.1. Normative References

[bidir] Lever, C., "Size-Limited Bi-directional Remote Procedure Call On Remote Direct Memory Access Transports", April 2016, <<http://www.ietf.org/id/draft-ietf-nfsv4-rpcrdma-bidirection-02.txt>>.

Work in progress.

[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>>.

[rfc5666bis]

Lever, C., Ed., Simpson, W., and T. Talpey, "Remote Direct Memory Access Transport for Remote Procedure Call", May 2016, <<http://www.ietf.org/id/draft-ietf-nfsv4-rfc5666bis-07.txt>>.



Work in progress.

[rpccrdmav2]

Lever, C., Ed. and D. Noveck, "RPC-over-RDMA Version Two", June 2016, <<http://www.ietf.org/id/draft-cel-nfsv4-rpccrdma-version-two-01.txt>>.

Work in progress.

## 9.2. Informative References

- [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 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 [[rfc5666bis](#)] and for his helpful review of and suggestions for this document.

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  
Hewlett Packard Enterprise  
165 Dascomb Road  
Andover, MA 01810  
USA

Phone: +1 781-572-8038  
Email: [davenoveck@gmail.com](mailto:davenoveck@gmail.com)

