Network File System Version 4
Internet-Draft
Obsoletes: 5666 (if approved)
Intended status: Standards Track
Expires: June 3, 2016

C. Lever, Ed.
Oracle
T. Talpey
Microsoft

December 1, 2015

Remote Direct Memory Access Transport for Remote Procedure Call
draft-ietf-nfsv4-rfc5666bis-00

Abstract

This document describes a protocol providing Remote Direct Memory Access (RDMA) as a new transport for Remote Procedure Call (RPC). The RDMA transport binding conveys the benefits of efficient, bulk-data transport over high-speed networks, while providing for minimal change to RPC applications and with no required revision of the application RPC protocol, or the RPC protocol itself. This document obsoletes RFC 5666.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document obsoletes RFC 5666, but makes no operational changes to RPC-over-RDMA Version One protocol on the wire. It is published to clarify ambiguous text that is subject to multiple interpretations, deprecate unimplemented RPC-over-RDMA Version One protocol elements, and introduce conventions to allow bi-directional RPC-over-RDMA operation.

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. RPC Over RDMA Transports

Remote Direct Memory Access (RDMA) [RFC5040] [RFC5041] [IB] is a technique for efficient movement of data between end nodes, which becomes increasingly compelling over high-speed transports. By directing data into destination buffers as it is sent on a network, and placing it via direct memory access by hardware, the double benefit of faster transfers and reduced host overhead is obtained.

Open Network Computing Remote Procedure Call (ONC RPC, or simply, RPC) [RFC5531] is a remote procedure call protocol that has been run over a variety of transports. Most RPC implementations today use UDP or TCP. RPC messages are defined in terms of an eXternal Data Representation (XDR) [RFC4506], which provides a canonical data representation across a variety of host architectures. An XDR data stream is conveyed differently on each type of transport. On UDP, RPC messages are encapsulated inside datagrams, while on a TCP byte stream, RPC messages are delineated by a record marking protocol. An RDMA transport also conveys RPC messages in a unique fashion that must be fully described if RPC implementations are to interoperate.

RDMA transports present new semantics unlike the behaviors of either UDP or TCP alone. They retain message delineations like UDP while also providing a reliable, sequenced data transfer like TCP. Also, they provide the new efficient, bulk-transfer service enabled by Remote Direct Memory Access. RDMA transports are therefore naturally viewed as a new transport type by RPC.

RDMA as a transport will benefit the performance of RPC protocols that move large "chunks" of data, since RDMA hardware excels at moving data efficiently between host memory and a high-speed network with little or no host CPU involvement. In this context, the Network File System (NFS) protocol, in all its versions [RFC1094] [RFC1813] [RFC7530] [RFC5661], is an obvious beneficiary of RDMA. A complete problem statement is discussed in [RFC5532], and related NFSv4 issues are discussed in [RFC5661]. Many other RPC-based protocols can also benefit.

Although the RDMA transport described here provides relatively transparent support for any RPC application, this document goes further in describing mechanisms that can optimize the use of RDMA with more active participation by the RPC application.

2. Changes Since RFC 5666

2.1. Changes To The Specification

The following alterations have been made to the RPC-over-RDMA Version One specification:

- Often implementers familiar with RDMA are not familiar with the mechanics of RPC, and vice versa. Section 2 has been expanded to introduce and explain key RPC, XDR, and RDMA terminology. These terms are now used consistently throughout the specification.

- Section 3 has been re-organized and split into sub-sections to facilitate locating specific requirements and definitions.

- Section 4 and 5 have been combined for clarity and to improve the organization of this information.

- The specification of the optional Connection Configuration Protocol has been removed from the specification, as there are no known implementations of the protocol.

- Sections discussing requirements for Upper Layer Bindings have been added.

- A section discussing RPC-over-RDMA protocol extensibility has been added.

2.2. Changes To The Protocol

The specific changes to the protocol are:

- Support for the Read-Read transfer model has been deprecated. Read-Read is a slower transfer model than Read-Write, thus implementers have chosen not to support it.

- Support for the RDMA_MSGP message type has been deprecated. It has no benefit for RPC programs that place bulk payload items in the middle of their argument or result lists, as is typical with NFSv4 COMPOUND RPCs [RFC7530]. It is also not beneficial when the inline threshold is significantly smaller than the system page size.

- The XDR definition of RPC-over-RDMA Version One has been updated (without on-the-wire changes) to align with the terms and concepts introduced in this specification.

- Specific requirements related to handling XDR round-up and abstract data types have been added.

- Clear guidance about Send and Receive buffer size has been added. This enables better decisions about when to provide and use the Reply chunk.

- A section specifying bi-directional RPC operation on RPC-over-RDMA has been added. This enables the NFSv4.1 backchannel [RFC5661] on RPC-over-RDMA Version One transports.

The protocol version number is not changed because the protocol specified in this document fully interoperates with implementations of the RPC-over-RDMA Version One protocol specified in [RFC5666].

3. Terminology

3.1. Remote Procedure Calls

This section introduces key elements of the Remote Procedure Call [RFC5531] and External Data Representation [RFC4506] protocols upon which RPC-over-RDMA Version One is constructed.

3.1.1. Upper Layer Protocols

Remote Procedure Calls are an abstraction used to implement the operations of an "upper layer protocol," sometimes referred to as a ULP. One example of such a protocol is the Network File System Version 4.0 [RFC7530].

3.1.2. Requesters And Responders

Like a local procedure call, every Remote Procedure Call has a set of "arguments" and a set of "results". A calling context is not allowed to proceed until the procedure's results are available to it. Unlike a local procedure call, the called procedure is executed remotely rather than in the local application's context.

The RPC protocol as described in [RFC5531] is fundamentally a message-passing protocol between one server and one or more clients. ONC RPC transactions are made up of two types of messages:

CALL Message
A CALL message, or "Call", requests work. A Call is designated by the value CALL in the message's msg_type field. An arbitrary unique value is placed in the message's xid field.

REPLY Message
A REPLY message, or "Reply", reports the results of work requested by a Call. A Reply is designated by the value REPLY in the message's msg_type field. The value contained in the message's xid field is copied from the Call whose results are being reported.

An RPC client endpoint, or "requester", serializes an RPC call's arguments and conveys them to a server endpoint via an RPC call message. This message contains an RPC protocol header, a header describing the requested upper layer operation, and all arguments.

The server endpoint, or "responder", deserializes the arguments and processes the requested operation. It then serializes the operation's results into another byte stream. This byte stream is conveyed back to the requester via an RPC reply message. This message contains an RPC protocol header, a header describing the upper layer reply, and all results. The requester deserializes the results and allows the original caller to proceed.

RPC-over-RDMA is a connection-oriented RPC transport. When a connection-oriented transport is used, ONC RPC client endpoints are responsible for initiating transport connections, while ONC RPC service endpoints wait passively for incoming connection requests.

3.1.3. External Data Representation

In a heterogenous environment, one cannot assume that all requesters and responders represent data the same way. RPC uses eXternal Data Representation, or XDR, to translate data types and serialize arguments and results. The XDR protocol encodes data independent of the endianness or size of host-native data types, allowing unambiguous decoding of data on the receiving end. RPC programs are specified by writing an XDR definition of their procedures, argument data types, and result data types.

XDR assumes that the number of bits in a byte (octet) and their order are the same on both endpoints and on the physical network. The smallest indivisible unit of XDR encoding is a group of four octets in little-endian order. XDR also flattens lists, arrays, and other abstract data types so they can be conveyed as a simple stream of bytes.

A serialized stream of bytes that is the result of XDR encoding is referred to as an "XDR stream." A sending endpoint encodes native data into an XDR stream and then transmits that stream to a receiver. A receiving endpoint decodes incoming XDR byte streams into its native data representation format.

The function of an RPC transport is to convey RPC messages, each encoded as a separate XDR stream, from one endpoint to another.

3.1.3.1. XDR Opaque Data

Sometimes a data item must be transferred as-is, without encoding or decoding. Such a data item is referred to as "opaque data." XDR encoding places opaque data items directly into an XDR stream without altering its content in any way.

Typically Upper Layer Protocols or applications manage any needed data translation in this case. Examples of opaque data items include the contents of files, and generic byte strings.

3.1.3.2. XDR Round-up

The number of octets in a variable-size data item precedes that item in the encoding stream. If the size of an encoded data item is not a multiple of four octets, octets containing zero are added to the end of the item so that the next encoded data item starts on a four-octet boundary. The encoded size of the item is not changed by the addition of the extra octets.

This technique is referred to as "XDR round-up," and the extra octets are referred to as "XDR padding". The content of XDR pad octets is ignored by receivers.

3.2. Remote Direct Memory Access

An RPC requester can be made more efficient if large RPC messages are transferred by a third party such as intelligent network interface hardware (data movement offload), and placed in the receiver's memory so that no additional adjustment of data alignment has to be made (direct data placement). Remote Direct Memory Access, or "RDMA" is a network transport technology that enables both optimizations.

3.2.1. Direct Data Placement

Very often, RPC implementations copy the contents of RPC messages into a buffer before being sent. An efficient RPC implementation can send bulk data without copying it into a separate send buffer first.

However, socket-based RPC implementations are often unable to receive data directly into its final place in memory. Receivers often need to copy incoming data to finish an RPC operation; sometimes, only to adjust data alignment.

In this document, "RDMA" refers to the physical mechanism an RDMA transport utilizes when moving data. Though it may not be optimal, before an RDMA transfer, the sender may still copy data into place. After an RDMA transfer, the receiver may copy that data again to its final destination.

This document uses the term "direct data placement" (or DDP) to refer specifically to an optimized data transfer where it is unnecessary for a receiving host's CPU to copy transferred data again after it has been received. Not all RDMA-based data transfer qualifies as Direct Data Placement, and DDP can be achieved using non-RDMA mechanisms.

3.2.2. RDMA Transport Requirements

The RPC-over-RDMA Version One protocol assumes the physical transport provides the following abstract operations. A more complete discussion of these operations is found in [RFC5040].

RDMA Send

The RDMA provider supports an RDMA Send operation with completion signaled at the receiver when data is placed in a pre-posted buffer. The amount of transferred data is limited only by the size of the receiver's buffer. Sends complete at the receiver in the order they were issued at the sender.

RDMA Receive

Receive endpoints pre-post enough RDMA Receive operations to catch incoming RDMA Send operations. To reduce the amount of memory that must remain pinned awaiting incoming Sends, receive buffers are limited in size and number. Flow-control to prevent overrunning receiver resources is provided by the upper layer protocol.

Registered Memory

All data moved via tagged RDMA operations is resident in registered memory at its destination. This protocol assumes that each segment of registered memory MUST be identified with a steering tag of no more than 32 bits and memory addresses of up to 64 bits in length.

RDMA Write

The RDMA provider supports an RDMA Write operation to directly place data in the receiver's buffer. An RDMA Write is initiated by the sender and completion is signaled at the sender. No completion is signaled at the receiver. The sender uses a steering tag, memory address, and length of the remote destination buffer.

RDMA Writes are not necessarily ordered with respect to one another, but are ordered with respect to RDMA Sends. A subsequent RDMA Send completion obtained at the receiver guarantees that prior RDMA Write data has been successfully placed in the receiver's memory.

RDMA Read

The RDMA provider supports an RDMA Read operation to directly place peer source data in the requester's buffer. An RDMA Read is initiated by the receiver and completion is signaled at the receiver. The receiver provides steering tags, memory addresses, and a length for the remote source and local destination buffers. Since the peer at the data source receives no notification of RDMA Read completion, there is an assumption that on receiving the data, the receiver will signal completion with an RDMA Send message, so that the peer can free the source buffers and the associated steering tags.

The RPC-over-RDMA Version One protocol is designed to be carried over RDMA transports that support the above abstract operations. This protocol conveys to the RPC peer information sufficient for that RPC peer to direct an RDMA layer to perform transfers containing RPC data and to communicate their result(s). For example, it is readily carried over RDMA transports such as Internet Wide Area RDMA Protocol (iWARP) [RFC5040] [RFC5041], or InfiniBand [IB].

4. Protocol Framework

4.1. Transfer Models

A "transfer model" designates which endpoint is responsible for performing RDMA Read and Write operations. To enable these operations, the peer endpoint first exposes segments of its memory to the endpoint performing the RDMA Read and Write operations.

Read-Read
Requesters expose their memory to the responder, and the responder exposes its memory to requesters. The responder employs RDMA Read operations to convey RPC arguments or whole RPC calls. Requesters employ RDMA Read operations to convey RPC results or whole RPC relies.

Write-Write
Requesters expose their memory to the responder, and the responder exposes its memory to requesters. Requesters employ RDMA Write operations to convey RPC arguments or whole RPC calls. The responder employs RDMA Write operations to convey RPC results or whole RPC relies.

Read-Write
Requesters expose their memory to the responder, but the responder does not expose its memory. The responder employs RDMA Read operations to convey RPC arguments or whole RPC calls. The responder employs RDMA Write operations to convey RPC results or whole RPC relies.

Write-Read
The responder exposes its memory to requesters, but requesters do not expose their memory. Requesters employ RDMA Write operations to convey RPC arguments or whole RPC calls. Requesters employ RDMA Read operations to convey RPC results or whole RPC relies.

[RFC5666] specifies the use of both the Read-Read and the Read-Write Transfer Model. All current RPC-over-RDMA Version One implementations use the Read-Write Transfer Model. Use of the Read-Read Transfer Model by RPC-over-RDMA Version One implementations is therefore deprecated. Other Transfer Models may be used by a future version of RPC-over-RDMA.

4.2. RPC-over-RDMA Framing

During transmission, the XDR stream containing an RPC message is preceded by an RPC-over-RDMA header. This header is analogous to the record marking used for RPC over TCP but is more extensive, since RDMA transports support several modes of data transfer.

All transfers of an RPC message begin with an RDMA Send that transfers an RPC-over-RDMA header and part or all of the accompanying RPC message. Because the size of what may be transmitted via RDMA Send is limited by the size of the receiver’s pre-posted buffers, the RPC-over-RDMA transport provides a number of methods to reduce the amount transferred via RDMA Send. Parts of RPC messages not transferred via RDMA Send are transferred using RDMA Read or RDMA Write operations.

RPC-over-RDMA framing replaces all other RPC framing (such as TCP record marking) when used atop an RPC-over-RDMA association, even when the underlying RDMA protocol may itself be layered atop a transport with a defined RPC framing (such as TCP).

It is however possible for RPC-over-RDMA to be dynamically enabled in the course of negotiating the use of RDMA via an Upper Layer Protocol exchange. Because RPC framing delimits an entire RPC request or reply, the resulting shift in framing must occur between distinct RPC messages, and in concert with the transport.

4.3. Flow Control

It is critical to provide RDMA Send flow control for an RDMA connection. RDMA receive operations can fail if a pre-posted receive buffer is not available to accept an incoming RDMA Send, and repeated occurrences of such errors can be fatal to the connection. This is a departure from conventional TCP/IP networking where buffers are allocated dynamically as part of receiving messages.

It is not practical to provide for fixed credit limits at the responder. Fixed limits scale poorly, since posted buffers are dedicated to the associated connection until consumed by receive operations. In addition, for protocol correctness, a responder must always be able to reply to requesters, whether or not the responder has posted buffers to accept more requests.

Therefore, flow control for RDMA Send operations is implemented as a simple request/grant protocol in the RPC-over-RDMA header associated with each RPC message. The RPC-over-RDMA header for RPC call messages contains a requested credit value for the responder, which MAY be dynamically adjusted by the caller to match its expected needs.

The RPC-over-RDMA header for RPC reply messages provides the granted result, which MAY have any value except it MUST NOT be zero when no in-progress operations are present at the responder, since such a value would result in deadlock. The value MAY be adjusted up or down at each opportunity to match the responder’s needs or policies.

The requester MUST NOT send unacknowledged requests in excess of this granted responder credit limit. If the limit is exceeded, the RDMA layer may signal an error, possibly terminating the connection. Even if an error does not occur, it is OPTIONAL that the responder handle the excess request(s). it MAY return an RPC error to the requester

Note that the never-zero requirement implies that an responder MUST always provide at least one credit to each connected requester from which no requests are outstanding. The requester would deadlock otherwise, unable to send another request.

While RPC calls complete in any order, the current flow control limit at the responder is known to the requester from the Send ordering properties. It is always the most recent responder-granted credit value minus the number of requests in flight.

Certain RDMA implementations may impose additional flow control restrictions, such as limits on RDMA Read operations in progress at the responder. Because these operations are outside the scope of this protocol, they are not addressed and SHOULD be provided for by other layers.

4.3.1. Initial Connection State

There are two operational parameters for each connection:

Credit Limit
The number of available receive buffers is a connection’s credit limit. The credit limit is advertised in the RPC-over-RDMA header in each RPC message, and can change during the lifetime of a connection.

Inline Threshold
The maximum RDMA message size that can be received is a connection’s "inline threshold." This is the size of the smallest posted receive buffer, though usually all of a connection’s receive buffers are the same size. Unlike the connection’s credit limit, the inline threshold value is not advertised to peers via the RPC-over-RDMA Version One protocol, and there is no provision for the inline threshold value to change during the lifetime of an RPC-over-RDMA Version One connection.

The longevity of a transport connection requires that sending endpoints respect the resource limits of peer receivers. However, when a connection is first established, peers cannot know how many receive buffers the other has, nor how large the buffers are.

To provide a basis for an initial exchange of RPC requests, each RPC-over-RDMA connection is assumed to provide a basic level of interoperability: the ability to exchange at least one RPC message at a time that is 1024 bytes in size. A responder MAY exceed this basic level of configuration, but a requester MUST NOT assume more than one credit is available, and MUST receive a valid reply from the responder carrying the actual number of available credits, prior to sending its next request.

In the absense of an exchange of buffer size information (such as the Connection Configuration Protocol described in [RFC5666]), senders MUST assume the receiver’s inline threshold is 1024 bytes. Implementations MUST support an inline threshold of 1024 bytes, but MAY support larger inline thresholds.

4.4. XDR Encoding With Chunks

On traditional RPC transports, XDR data items in an RPC message are encoded as a contiguous sequence of bytes for network transmission. However, in the case of an RDMA transport, during XDR encoding it can be determined that (for instance) an opaque byte array is large enough to be moved via an RDMA Read or RDMA Write operation.

RPC-over-RDMA Version One provides a mechanism for moving part an RPC message via a separate RDMA data transfer. A contiguous piece of an XDR stream that is split out and moved via a separate RDMA operation is known as a "chunk." The sender removes the chunk data out from the XDR stream conveyed via RDMA Send, and the receiver inserts it before handing the reconstructed stream to the Upper Layer.

4.4.1. DDP-Eligibility

Only an XDR data item that might benefit from Direct Data Placement should be moved to a chunk. The eligibility of specific XDR data items to be moved as a chunk, as opposed to being left in the XDR stream, is not specified by this document. The Upper Layer Protocol MUST determine which items in its XDR definition are allowed to use Direct Data Placement. Therefore an additional specification is needed that describes how an Upper Layer Protocol enables Direct Data Placement. The set of requirements for a ULP to use an RDMA transport is known as an "Upper Layer Binding" specification, or ULB.

An Upper Layer Binding states which specific individual XDR data items in an Upper Layer Protocol MAY be transferred via Direct Data Placement. This document will refer to such XDR data items as "DDP-eligible". All other XDR data items MUST NOT be placed in a chunk. RPC-over-RDMA Version One uses RDMA Read and Write operations to transfer DDP-eligible data that has been placed in chunks.

The details and requirements for Upper Layer Bindings are discussed in full in Section 6.

4.4.2. RDMA Segments

When encoding an RPC message that contains a DDP-eligible data item, the RPC-over-RDMA transport does not place the item into the RPC message’s XDR stream. Instead, it records in the RPC-over-RDMA header the address and size of the memory region containing the data item. The requester sends this information for DDP-eligible data items in both RPC calls and replies. The responder uses this information to initiate RDMA Read and Write operations on the memory regions.

An "RDMA segment", or just "segment", is an RPC-over-RDMA header data object that contain the precise co-ordinates of a contiguous memory region that is to be conveyed via one or more RDMA Read or RDMA Write operations. The following fields are contained in a segment:

Handle
   Steering tag or handle obtained when the segment’s memory is registered for RDMA. Sometimes known as an R_key.

Length
   The length of the segment in bytes.

Offset
   The offset or beginning memory address of the segment.

See [RFC5040] for further discussion of the meaning of these fields.

4.4.3. Chunks

A "chunk" refers to a portion of XDR stream data that is moved via RDMA Read or Write operations. Chunk data is removed from the sender’s XDR stream, transferred by separate RDMA operations, and then re-inserted into the receiver’s XDR stream.

Each chunk consists of one or more RDMA segments. Each segment represents a single contiguous piece of that chunk.

Except in special cases, a chunk contains exactly one XDR data item. This makes it straightforward to remove chunks from an XDR stream without affecting XDR alignment.

```
      +----------------+ +----------------+------------------
      | RPC-over-RDMA  | |                |
      |    header w/   | |   RPC Header   | Non-chunk args/results
      |    segments    | |                |
      +----------------+ +----------------+------------------
               |
               +-> Chunk A
               +-> Chunk B
               +-> Chunk C
                    . . .
```

Block diagram of an RPC-over-RDMA message

Not every message has chunks associated with it. The structure of the RPC-over-RDMA header is covered in Section 5.

4.4.3.1. Counted Arrays

If a chunk is to move a counted array data type, the count of array elements MUST remain in the XDR stream, while the array elements MUST be moved to the chunk. For example, when encoding an opaque byte array as a chunk, the count of bytes stays in the XDR stream, while the bytes in the array are removed from the XDR stream and transferred via the chunk. Any byte count left in the XDR stream MUST match the sum of the lengths of the segments making up the chunk. If they do not agree, an RPC protocol encoding error results.

Individual array elements appear in the chunk in their entirety. For example, when encoding an array of arrays as a chunk, the count of items in the enclosing array stays in the XDR stream, but each enclosed array, including its item count, is transferred as part of the chunk.

4.4.3.2. Optional-data And Unions

If a chunk is to move an optional-data data type, the "is present" field MUST remain in the XDR stream, while the data, if present, MUST be moved to the chunk.

A union data type should never be made DDP-eligible, but one or more of its arms may be DDP-eligible.

4.4.4. Read Chunks

A "Read chunk" represents an XDR data item that is to be pulled from the requester to the responder using RDMA Read operations.

A Read chunk is a list of one or more RDMA segments. Each segment in a Read chunk has an additional Position field.

Position
For data that is to be encoded, the byte offset in the RPC message XDR stream where the receiver re-inserts the chunk data. The byte offset MUST be computed from the beginning of the RPC message, not the beginning of the RPC-over-RDMA header. All segments belonging to the same Read chunk have the same value in their Position field.

While constructing the RPC call, the requester registers memory regions containing data in Read chunks. It advertises these chunks in the RPC-over-RDMA header of the RPC call.

After receiving the RPC call via an RDMA Send operation, the responder transfers the chunk data from the requester using RDMA Read operations. The responder reconstructs the transferred chunk data by concatenating the contents of each segment, in list order, into the RPC call XDR stream. The first segment begins at the XDR position in the Position field, and subsequent segments are concatenated afterwards until there are no more segments left at that XDR Position.

4.4.4.1. Read Chunk Round-up

XDR requires each encoded data item to start on four-byte alignment. When an odd-length data item is marshaled, its length is encoded literally, while the data is padded so the next data item can start on a four-byte boundary in the XDR stream. Receivers ignore the content of the pad bytes.

Data items remaining in the XDR stream must all adhere to the above padding requirements. When a Read chunk is removed from an XDR stream, the requester MUST remove any needed XDR padding for that chunk as well. Alignment of the items remaining in the stream is unaffected.

The length of a Read chunk is the sum of the lengths of the segments that comprise it. If this sum is not a multiple of four, the requester MAY choose to send a Read chunk without any XDR padding. The responder MUST be prepared to provide appropriate round-up in its reconstructed XDR stream if the requester provides no actual round-up in a Read chunk.

The Position field in read segments indicates where the containing Read chunk starts in the RPC message XDR stream. The value in this field MUST be a multiple of four. Moreover, all segments in the same

Read chunk share the same Position value, even if one or more of the segments have a non-four-byte aligned length.

4.4.4.2. Decoding Read Chunks

XDR decoding moves data from an XDR stream into a data structure provided by an RPC application. Where elements of the destination data structure are buffers or strings, the RPC application can either pre-allocate storage to receive the data, or leave the string or buffer fields null and allow the XDR decode stage of RPC processing to automatically allocate storage of sufficient size.

When decoding a message from an RDMA transport, the receiver first decodes the chunk lists from the RPC-over-RDMA header, then proceeds to decode the body of the RPC message. Whenever the XDR offset in the decode stream matches that of a Read chunk, the transport initiates an RDMA Read to bring over the chunk data into locally registered memory for the destination buffer.

When processing an RPC request, the responder acknowledges its completion of use of the source buffers by simply replying to the requester. The requester may then free all source buffers advertised by the request.

4.4.5. Write Chunks

A "Write chunk" represents an XDR data item that is to be pushed from the responder to the requester using RDMA Write operations.

A Write chunk is an array of one or more RDMA segments. Segments in a Write chunk do not have a Position field because Write chunks are provided by a requester long before the responder prepares the reply XDR stream.

While constructing the RPC call, the requester also sets up memory regions to catch DDP-eligible reply data. The requester provides as many segments as needed to accommodate the largest possible size of the data item in each Write chunk.

The responder transfers the chunk data to the requester using RDMA Write operations. The responder copies the responder’s Write chunk segments into the RPC-over-RDMA header to be sent with the reply. The responder updates the segment length fields to reflect the actual amount of data that is being returned in the chunk. The updated length of a Write chunk segment MAY be zero if the segment was not filled by the responder. However the responder MUST NOT change the number of segments in the Write chunk.

The responder then sends the RPC reply via an RDMA Send operation. After receiving the RPC reply, the requester reconstructs the transferred data by concatenating the contents of each segment, in array order, into RPC reply XDR stream.

4.4.5.1. Unused Write Chunks

There are occasions when a requester provides a Write chunk but the responder does not use it. For example, an Upper Layer Protocol may have a union result where some arms of the union contain a DDP-eligible data item, and other arms do not. To return an unused Write chunk, the responder MUST set the length of all segments in the chunk to zero.

Unused write chunks, or unused bytes in write chunk segments, are not returned as results and their memory is returned to the Upper Layer as part of RPC completion. However, the RPC layer MUST NOT assume that the buffers have not been modified.

4.4.5.2. Write Chunk Round-up

XDR requires each encoded data item to start on four-byte alignment. When an odd-length data item is marshaled, its length is encoded literally, while the data is padded so the next data item can start on a four-byte boundary in the XDR stream. Receivers ignore the content of the pad bytes.

Data items remaining in the XDR stream must all adhere to the above padding requirements. When a Write chunk is removed from an XDR stream, the requester MUST remove any needed XDR padding for that chunk as well. Alignment of the items remaining in the stream is unaffected.

The length of a Write chunk is the sum of the lengths of the segments that comprise it. If this sum is not a multiple of four, the responder MAY choose not to write XDR padding. The requester does not know the actual length of a Write chunk when it is prepared, but it SHOULD provide enough segments to accommodate any needed XDR padding. The requester MUST be prepared to provide appropriate round-up in its reconstructed XDR stream if the responder provides no actual round-up in a Write chunk.

4.5. Data Exchange

In summary, there are three mechanisms for moving data between requester and responder.

Inline

Data is moved between requester and responder via an RDMA Send operation.

RDMA Read
Data is moved between requester and responder via an RDMA Read operation. Address and offset are obtained from a Read chunk in the requester’s RPC call message.

RDMA Write
Data is moved from responder to requester via an RDMA Write operation. Address and offset are obtained from a Write chunk in the requester’s RPC call message.

Many combinations are possible. For instance, an RPC call may contain some inline data along with Read or Write chunks. The reply to that call may have chunks that the responder RDMA Writes back to the requester. The following diagrams illustrate RPC calls that use these methods to move RPC message data.

```
     Requester                              Responder
         |              RPC Call                |
    Send |   ------------------------------>    |
         |                                      |
         |              RPC Reply               |
         |   <------------------------------    | Send
```

An RPC with no chunks in the call or reply messages

```
    Requester                              Responder
        |        RPC Call + Write chunks       |
   Send |   ------------------------------>    |
        |                                      |
        |             Chunk 1                  |
        |   <------------------------------    | Write
        |                :                     |
        |             Chunk n                  |
        |   <------------------------------    | Write
        |                                      |
        |             RPC Reply                |
        |   <------------------------------    | Send
```

An RPC with write chunks in the call message

In the presence of write chunks, RDMA ordering guarantees that all data in the RDMA Write operations has been placed in memory prior to the requester’s RPC reply processing.

```
       Requester                             Responder
           |        RPC Call + Read chunks      |
      Send |   ------------------------------>   |
           |                                     |
           |               Chunk 1               |
           |   +------------------------------   | Read
           |   v----------------------------->   |
           |                  :                  |
           |               Chunk n               |
           |   +------------------------------   | Read
           |   v----------------------------->   |
           |                                     |
           |               RPC Reply             |
           |   <------------------------------   | Send
```

An RPC with read chunks in the call message

```
       Requester                             Responder
           |   RPC Call + Read and Write chunks  |
      Send |   ------------------------------>   |
           |                                     |
           |             Read chunk 1            |
           |   +------------------------------   | Read
           |   v----------------------------->   |
           |                  :                  |
           |             Read chunk n            |
           |   +------------------------------   | Read
           |   v----------------------------->   |
           |                                     |
           |             Write chunk 1           |
           |   <------------------------------   | Write
           |                  :                  |
           |             Write chunk n           |
           |   <------------------------------   | Write
           |                                     |
           |               RPC Reply             |
           |   <------------------------------   | Send
```

An RPC with read and write chunks in the call message

4.6. Message Size

The receiver of RDMA Send operations is required by RDMA to have previously posted one or more adequately sized buffers (see Section 4.3.1). Memory savings can be achieved on both requesters and responders by leaving the inline threshold small.

4.6.1. Short Messages

RPC messages are frequently smaller than the connection’s inline threshold.

For example, the NFS version 3 GETATTR request is only 56 bytes: 20 bytes of RPC header, plus a 32-byte file handle argument and 4 bytes for its length. The reply to this common request is about 100 bytes.

Since all RPC messages conveyed via RPC-over-RDMA require an RDMA Send operation, the most efficient way to send an RPC message that is smaller than the connection’s inline threshold is to append its XDR stream directly to the buffer carrying the RPC-over-RDMA header. An RPC-over-RDMA header with a small RPC call or reply message immediately following is transferred using a single RDMA Send operation. No RDMA Read or Write operations are needed.

4.6.2. Chunked Messages

If DDP-eligible data items are present in an RPC message, a sender MAY remove them from the RPC message, and use RDMA Read or Write operations to move that data. The RPC-over-RDMA header with the shortened RPC call or reply message immediately following is transferred using a single RDMA Send operation. Removed DDP-eligible data items are conveyed using RDMA Read or Write operations using additional information provided in the RPC-over-RDMA header.

4.6.3. Long Messages

When an RPC message is larger than the connection’s inline threshold and the Upper Layer Binding does not identify any DDP-eligible data items in the requested operation that may be moved separately, the RDMA transport MUST use RDMA Read and Write operations to convey the whole RPC message. This mechanism is referred to as a "Long Message."

To send an RPC message as a Long Message, the sender conveys only the RPC-over-RDMA header with an RDMA Send operation. The RPC message itself is not included in the Send buffer. Instead, the requester provides chunks that the responder uses to move the whole RPC message.

Long RPC call

To handle an RPC request using a Long Message, the requester provides a special Read chunk that contains the RPC call’s XDR stream. Every segment in this Read chunk MUST contain zero in its Position field. This chunk is known as a "Position Zero Read chunk."

Long RPC reply

To handle an RPC reply using a Long Message, the requester provides a single special Write chunk, known as the "Reply chunk", that contains the RPC reply’s XDR stream. The requester sizes the Reply chunk to accommodate the largest possible expected reply for that Upper Layer operation.

Though the purpose of a Long Message is to handle large RPC messages, requesters MAY use a Long Message at any time to convey an RPC call. Responders SHOULD use a Long Message whenever a Reply chunk has been provided by a requester. Both types of special chunk MAY be present in the same RPC message.

Because these special chunks contain a whole RPC message, any XDR data item MAY appear in one of these special chunks without regard to its DDP-eligibility. DDP-eligible data items MAY be removed from these special chunks and conveyed via normal chunks, but non-eligible data items MUST NOT appear in normal chunks.

5. RPC-over-RDMA In Operation

An RPC-over-RDMA Version One header precedes all RPC messages conveyed across an RDMA transport. This header includes a copy of the message’s transaction ID, data for RDMA flow control credits, and lists of memory addresses used for RDMA Read and Write operations. All RPC-over-RDMA header content MUST be XDR encoded.

RPC message layout is unchanged from that described in [RFC5531] except for the possible removal of data items that are moved by RDMA Read or Write operations. If an RPC message (along with its RPC-over-RDMA header) is larger than the connection’s inline threshold even after any large chunks are removed, then the RPC message MAY be moved separately as a chunk, leaving just the RPC-over-RDMA header in the RDMA Send.

5.1. Fixed Header Fields

The RPC-over-RDMA header begins with four fixed 32-bit fields that MUST be present and that control the RDMA interaction including RDMA-specific flow control. These four fields are:

5.1.1. Transaction ID (XID)

The XID generated for the RPC call and reply. Having the XID at a fixed location in the header makes it easy for the receiver to establish context as soon as the message arrives. This XID MUST be the same as the XID in the RPC header. The receiver MAY perform its processing based solely on the XID in the RPC-over-RDMA header, and thereby ignore the XID in the RPC header, if it so chooses.

5.1.2. Version number

For RPC-over-RDMA Version One, this field MUST contain the value 1 (one). Further discussion of protocol extensibility can be found in Section 9.

5.1.3. Flow control credit value

When sent in an RPC call message, the requested credit value is provided. When sent in an RPC reply message, the granted credit value is returned. RPC calls SHOULD NOT be sent in excess of the currently granted limit. Further discussion of flow control can be found in Section 4.3.

5.1.4. Message type

- RDMA_MSG = 0 indicates that chunk lists and an RPC message follow. The format of the chunk lists is discussed below.

- RDMA_NOMSG = 1 indicates that after the chunk lists there is no RPC message. In this case, the chunk lists provide information to allow the responder to transfer the RPC message using RDMA Read or Write operations.

- RDMA_MSGP = 2 is reserved, and no longer used.

- RDMA_DONE = 3 is reserved, and no longer used.

- RDMA_ERROR = 4 is used to signal a responder-detected error in RDMA chunk encoding.

For a message of type RDMA_MSG, the four fixed fields are followed by the Read and Write lists and the Reply chunk (though any or all three MAY be marked as not present), then an RPC message, beginning with its XID field. The Send buffer holds two separate XDR streams: the first XDR stream contains the RPC-over-RDMA header, and the second XDR stream contains the RPC message.

For a message of type RDMA_NOMSG, the four fixed fields are followed by the Read and Write chunk lists and the Reply chunk (though any or all three MAY be marked as not present). The Send buffer holds one XDR stream which contains the RPC-over-RDMA header.

For a message of type RDMA_ERROR, the four fixed fields are followed by formatted error information.

The above content (the fixed fields, the chunk lists, and the RPC message, when present) MUST be conveyed via a single RDMA Send operation. A gather operation on the Send can be used to marshal the separate RPC-over-RDMA header, the chunk lists, and the RPC message itself. However, the total length of the gathered send buffers cannot exceed the peer’s inline threshold.

5.2. Chunk Lists

The chunk lists in an RPC-over-RDMA Version One header are three XDR optional-data fields that MUST follow the fixed header fields in RDMA_MSG and RDMA_NOMSG type messages. Read Section 4.19 of [RFC4506] carefully to understand how optional-data fields work. Examples of XDR encoded chunk lists are provided in Section 13.1 to aid understanding.

5.2.1. Read List

Each RPC-over-RDMA Version One header has one "Read list." The Read list is a list of zero or more Read segments, provided by the requester, that are grouped by their Position fields into Read chunks. Each Read chunk advertises the locations of data the responder is to pull via RDMA Read operations. The requester SHOULD sort the chunks in the Read list in Position order.

Via a Position Zero Read Chunk, a requester may provide part or all of an entire RPC call message as the first chunk in this list.

The Read list MAY be empty if the RPC call has no argument data that is DDP-eligible and the Position Zero Read Chunk is not being used.

5.2.2. Write List

Each RPC-over-RDMA Version One header has one "Write list." The Write list is a list of zero or more Write chunks, provided by the requester. Each Write chunk is an array of RDMA segments, thus the Write list is a list of counted arrays. Each Write chunk advertises receptacles for DDP-eligible data to be pushed by the responder.

When a Write list is provided for the results of the RPC call, the responder MUST provide any corresponding data via RDMA Write to the memory referenced in the chunk's segments. The Write list MAY be empty if the RPC operation has no DDP-eligible result data.

When multiple Write chunks are present, the responder fills in each Write chunk with a DDP-eligible result until either there are no more results or no more Write chunks. An Upper Layer Binding MUST determine how Write list entries are mapped to procedure arguments for each Upper Layer procedure. For details, see Section 6.

The RPC reply conveys the size of result data by returning the Write list to the requester with the lengths rewritten to match the actual transfer. Decoding the reply therefore performs no local data transfer but merely returns the length obtained from the reply.

Each decoded result consumes one entry in the Write list, which in turn consists of an array of RDMA segments. The length of a Write chunk is therefore the sum of all returned lengths in all segments comprising the corresponding list entry. As each Write chunk is decoded, the entire entry is consumed.

5.2.3. Reply Chunk

Each RPC-over-RDMA Version One header has one "Reply Chunk." The Reply Chunk is a Write chunk, provided by the requester. The Reply Chunk is a single counted array of RDMA segments. A responder MAY convey part or all of an entire RPC reply message in this chunk.

A requester provides the Reply chunk whenever it predicts the responder's reply might not fit in an RDMA Send operation. A requester MAY choose to provide the Reply chunk even when the responder can return only a small reply.

5.3. Forming Messages

5.3.1. Short Messages

A Short Message without chunks is contained entirely within a single RDMA Send Operation. Since the RPC call message immediately follows the RPC-over-RDMA header in the send buffer, the requester MUST set the message type to RDMA_MSG.

```
   <------------------ RPC-over-RDMA header --------------->
   +--------+---------+---------+------------+-------------+ +----------
   |        |         |         |            |     NULL    | | Whole
   |  XID   | Version | Credits |  RDMA_MSG  | Chunk Lists | |  RPC
   |        |         |         |            |             | | Message
   +--------+---------+---------+------------+-------------+ +----------
```

5.3.2. Chunked Messages

A Chunked Message is similar to a Short Message, but the RPC message does not contain the chunk data. Since the RPC call message immediately follows the RPC-over-RDMA header in the send buffer, the requester MUST set the message type to RDMA_MSG.

```
   <------------------ RPC-over-RDMA header --------------->
   +--------+---------+---------+------------+-------------+ +----------
   |        |         |         |            |             | | Modified
   |  XID   | Version | Credits |  RDMA_MSG  | Chunk Lists | |   RPC
   |        |         |         |            |             | | Message
   +--------+---------+---------+------------+-------------+ +----------
                                                |
                                                |  +----------
                                                |  |
                                                +->| Chunks
                                                   |
                                                   +----------
```

5.3.3. Long Call Messages

To send a Long Call Message, the requester registers the memory containing the RPC call message and adds a chunk to the Read List at Position Zero. Since the RPC call message does not follow the RPC-over-RDMA header in the send buffer, the requester MUST set the message type to RDMA_NOMSG.

```
      <------------------ RPC-over-RDMA header --------------->
      +--------+---------+---------+------------+-------------+
      |        |         |         |            |             |
      |  XID   | Version | Credits | RDMA_NOMSG | Chunk Lists |
      |        |         |         |            |             |
      +--------+---------+---------+------------+-------------+
                                                   |
                                                   |  +----------
                                                   |  | RPC Call
                                                   +->|
                                                      | Message
                                                      +----------
```

If a responder gets an RPC-over-RDMA header with a message type of RDMA_NOMSG and finds an initial Read list entry with a zero XDR position, it allocates a registered buffer and issues an RDMA Read of the RPC message into it. The responder then proceeds to XDR decode the RPC message as if it had received it with the Send data. Further decoding may issue additional RDMA Reads to bring over additional chunks.

```
       Requester                             Responder
           |        RDMA-over-RPC Header         |
      Send |   ------------------------------>   |
           |                                     |
           |          Long RPC Call Msg          |
           |   <------------------------------   | Read
           |   ------------------------------>   |
           |                                     |
           |         RDMA-over-RPC Reply         |
           |   <------------------------------   | Send
```

A long call RPC with request supplied via RDMA Read

5.3.4. Long Reply Messages

To send a Long Reply Message, the requester MAY register a large buffer into which the responder can write an RPC reply. This buffer is passed to the responder in the RPC call message as the Reply chunk.

If the responder’s reply message is too long to return with an RDMA Send operation, even after Write chunks are removed, then the responder performs an RDMA Write of the RPC reply message into the buffer indicated by the Reply chunk. Since the RPC reply message does not follow the RPC-over-RDMA header in the send buffer, the responder MUST set the message type to RDMA_NOMSG.

```
      <------------------ RPC-over-RDMA header --------------->
      +--------+---------+---------+------------+-------------+
      |        |         |         |            |             |
      |  XID   | Version | Credits | RDMA_NOMSG | Chunk Lists |
      |        |         |         |            |             |
      +--------+---------+---------+------------+-------------+
                                                   |
                                                   |  +----------
                                                   |  | RPC Reply
                                                   +->|
                                                      | Message
                                                      +----------


       Requester                             Responder
           |      RPC Call with Reply chunk      |
      Send |   ------------------------------>   |
           |                                     |
           |          Long RPC Reply Msg         |
           |   <------------------------------   | Write
           |         RDMA-over-RPC Header        |
           |   <------------------------------   | Send
```

An RPC with long reply returned via RDMA Write

The use of RDMA Write to return long replies requires that the requester anticipates a long reply and has some knowledge of its size so that an adequately sized buffer can be allocated. Typically the Upper Layer Protocol can limit the size of RPC replies appropriately.

It is possible for a single RPC procedure to employ both a long call for its arguments and a long reply for its results. However, such an operation is atypical, as few upper layers define such exchanges.

5.4. Memory Registration

RDMA requires that all data be transferred between registered memory regions at the source and destination. All protocol headers as well as separately transferred data chunks use registered memory. Since the cost of registering and de-registering memory can be a large proportion of the RDMA transaction cost, it is important to minimize registration activity. This is easily achieved within RPC-controlled memory by allocating chunk list data and RPC headers in a reusable way from pre-registered pools.

Data chunks transferred via RDMA Read and Write MAY occupy memory that persists outside the bounds of the RPC transaction. Hence, the default behavior of an RPC-over-RDMA transport is to register and invalidate these chunks on every RPC transaction. The requester transport implementation must ensure that these memory regions are properly fenced from the responder before allowing Upper Layer access to the data contained in them.

The interface by which an upper-layer implementation communicates the eligibility of a data item locally to RPC for chunking is out of scope for this specification. Depending on the implementation and constraints imposed by Upper Layer Bindings, it is possible to implement an RPC chunking facility that is transparent to upper layers. However, such implementations may lead to inefficiencies, either because they require the RPC layer to perform expensive registration and de-registration of memory "on the fly", or they may require using RDMA chunks in reply messages, along with the resulting additional handshaking with the RPC-over-RDMA peer. However, these issues are internal and generally confined to the local interface between RPC and its upper layers, one in which implementations are free to innovate. The only requirement is that the resulting RPC-over-RDMA protocol sent to the peer is valid for the upper layer.

The choice of which memory registration strategies to employ is left to requester and responder implementers. To support the widest array of RDMA implementations, as well as the most general steering tag scheme, an Offset field is included in each segment.

While zero-based offset schemes are available in many RDMA implementations, their use by RPC requires individual registration of each segment. For such implementations, this can be a significant overhead. By providing an offset in each chunk, many pre-registration or region-based registrations can be readily supported. By using a single, universal chunk representation, the RPC-over-RDMA protocol implementation is simplified to its most general form.

5.5. Handling Errors

When a peer receives an RPC-over-RDMA message, it MUST perform basic validity checks on the header and chunk contents. If such errors are detected in the request, an RDMA_ERROR reply MUST be generated.

Two types of errors are defined, version mismatch and invalid chunk format.

- When a responder detects an RPC-over-RDMA header version that it does not support (currently this document defines only Version One), it replies with an error code of ERR_VERS, and provides the low and high inclusive version numbers it does, in fact, support. The version number in this reply MUST be any value otherwise valid at the receiver.

- When a responder detects other decoding errors in the header or chunks, one of the following errors MUST be returned: either an RPC decode error such as RPC_GARBAGEARGS, or the RPC-over-RDMA error code ERR_CHUNK.

When a requester cannot parse a responder’s reply, the requester SHOULD drop the RPC request and return an error to the application to prevent retransmission of an operation that can never complete.

A requester might not provide a responder with enough resources to reply. For example, if a requester’s receive buffer is too small, the responder’s Send operation completes with a Local Length Error, and the connection is dropped. Or, if the requester’s Reply chunk is too small to accommodate the whole RPC reply, the responder can tell as it is constructing the reply. The responder SHOULD send a reply with RDMA_ERROR to signal to the requester that no RPC-level reply is possible, and the XID should not be retried.

It is assumed that the link itself will provide some degree of error detection and retransmission. iWARP’s Marker PDU Aligned (MPA) layer (when used over TCP), Stream Control Transmission Protocol (SCTP), as well as the InfiniBand link layer all provide Cyclic Redundancy Check (CRC) protection of the RDMA payload, and CRC-class protection is a general attribute of such transports.

Additionally, the RPC layer itself can accept errors from the link level and recover via retransmission. RPC recovery can handle complete loss and re-establishment of the link. The details of reporting and recovery from RDMA link layer errors are outside the scope of this protocol specification.

See Section 10 for further discussion of the use of RPC-level integrity schemes to detect errors and related efficiency issues.

5.6. XDR Language Description

Code components extracted from this document must include the following license boilerplate.

```
<CODE BEGINS>
```

```
      /*
       * Copyright (c) 2010, 2015 IETF Trust and the persons
       * identified as authors of the code.  All rights reserved.
       *
       * The authors of the code are:
       * B. Callaghan, T. Talpey, and C. Lever.
       *
       * 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.
       */

      struct rpcrdma1_segment {
              uint32 rdma_handle;
              uint32 rdma_length;
              uint64 rdma_offset;
      };
      struct rpcrdma1_read_segment {
              uint32                  rdma_position;
              struct rpcrdma1_segment rdma_target;
      };

      struct rpcrdma1_read_list {
              struct rpcrdma1_read_segment rdma_entry;
              struct rpcrdma1_read_list    *rdma_next;
      };

      struct rpcrdma1_write_chunk {
              struct rpcrdma1_segment rdma_target<>;
      };

      struct rpcrdma1_write_list {
              struct rpcrdma1_write_chunk rdma_entry;
              struct rpcrdma1_write_list  *rdma_next;
      };

      struct rpcrdma1_msg {
              uint32        rdma_xid;
              uint32        rdma_vers;
              uint32        rdma_credit;
              rpcrdma1_body rdma_body;
      };

      enum rpcrdma1_proc {
              RDMA_MSG = 0,
              RDMA_NOMSG = 1,
              RDMA_MSGP = 2,  /* Reserved */
              RDMA_DONE = 3,  /* Reserved */
              RDMA_ERROR = 4
      };

      struct rpcrdma1_chunks {
              struct rpcrdma1_read_list   *rdma_reads;
              struct rpcrdma1_write_list  *rdma_writes;
              struct rpcrdma1_write_chunk *rdma_reply;
      };

      enum rpcrdma1_errcode {
              RDMA_ERR_VERS = 1,
              RDMA_ERR_CHUNK = 2
      };

      union rpcrdma1_error switch (rpcrdma1_errcode err) {
              case RDMA_ERR_VERS:
                uint32 rdma_vers_low;
                uint32 rdma_vers_high;
              case RDMA_ERR_CHUNK:
                void;
      };

      union rdma_body switch (rpcrdma1_proc proc) {
              case RDMA_MSG:
              case RDMA_NOMSG:
                rpcrdma1_chunks rdma_chunks;
              case RDMA_MSGP:
                uint32          rdma_align;
                uint32          rdma_thresh;
                rpcrdma1_chunks rdma_achunks;
              case RDMA_DONE:
                void;
              case RDMA_ERROR:
                rpcrdma1_error rdma_error;
      };

   <CODE ENDS>
```

5.7. Deprecated Protocol Elements

5.7.1. RDMA_MSGP

Implementers of RPC-over-RDMA Version One have neglected to make use of the RDMA_MSGP message type. Therefore RDMA_MSGP is deprecated.

Senders SHOULD NOT send RDMA_MSGP type messages. Receivers MUST treat received RDMA_MSGP type messages as they would treat RDMA_MSG type messages. The additional alignment information is an optimization hint that may be ignored.

5.7.2. RDMA_DONE

Because implementations of RPC-over-RDMA Version One do not use the Read-Read transfer model, there should never be any need to send an RDMA_DONE type message. Therefore RDMA_DONE is deprecated.

Receivers MUST drop RDMA_DONE type messages without additional processing.

6. Upper Layer Binding Specifications

Each RPC program and version tuple that operates on an RDMA transport MUST have an Upper Layer Binding specification. A ULB may be part of another protocol specification, or it may be a stand-alone document, similar to [RFC5667].

A ULB specification MUST provide four important pieces of information:

- Which XDR data items in the RPC program are eligible for Direct Data Placement

- How a responder utilizes chunks provided in a Write list

- How DDP-eligibility violations are reported to peers

- An rpcbind port assignment for operation of the RPC program on RDMA transports

6.1. Determining DDP-Eligibility

A DDP-eligible XDR data item is one that MAY be moved in a chunk. All other XDR data items MUST NOT be moved in a chunk that is part of a Short or Chunked Message, nor as a separate chunk in a Long Message.

Only an XDR data item that might benefit from Direct Data Placement should be transferred in a chunk. An Upper Layer Binding specification should consider an XDR data item for DDP-eligibility if the data item can be larger than a Send buffer, and at least one of the following:

- Is sensitive to page alignment (eg. it would require pullup on the receiver before it can be used)

- Is not translated or marshaled when it is XDR encoded (eg. an opaque array)

- Is not immediately used by applications (eg. is part of data backup or replication)

The Upper Layer Protocol implementation or the RDMA transport implementation decide when to move a DDP-eligible data item into a chunk instead of leaving the item in the RPC message’s XDR stream. The interface by which an Upper Layer implementation communicates the chunk eligibility of a data item locally to the RPC transport is out of scope for this specification. The only requirement is that the resulting RPC-over-RDMA protocol sent to the peer is valid for the Upper Layer.

The XDR language definition of DDP-eligible data items is not decorated in any way.

It is the responsibility of the protocol's Upper Layer Binding to specify DDP-eligibity rules so that if a DDP-eligible XDR data item is embedded within another, only one of these two objects is to be represented by a chunk. This ensures that the mapping from XDR position to the XDR object represented is unambiguous.

An Upper Layer Binding is considered ready to publish when:

- Every XDR data type in the protocol has been considered for DDP-eligibility

- Long Messages

6.2. Write List Ordering

Place holder

An Upper Layer Binding MUST determine how Write list entries are mapped to procedure arguments for each Upper Layer procedure.

6.3. DDP-Eligibility Violation

If the Upper Layer on a receiver is not aware of the presence and operation of an RPC-over-RDMA transport under it, it could be challenging to discover when a sender has violated an Upper Layer Binding rule.

If a violation does occur, RFC 5666 does not define an unambiguous mechanism for reporting the violation. The violation of Binding rules is an Upper Layer Protocol issue, but it is likely that there is nothing the Upper Layer can do but reply with the equivalent of BAD XDR.

When an erroneously-constructed reply reaches a requester, there is no recourse but to drop the reply, and perhaps the transport connection as well.

Policing DDP-eligibility must be done in co-operation with the Upper Layer Protocol by its receive endpoint implementation.

It is the Upper Layer Binding's responsibility to specify how a responder must reply if a requester violates a DDP-eligibilty rule. The Binding specification should provide similar guidance for requesters about handling invalid RPC-over-RDMA replies.

6.4. Other Binding Information

An Upper Layer Binding may recommend inline threshold values for RPC-over-RDMA Version One connections bearing that Upper Layer Protocol. However, note that RPC-over-RDMA connections can be shared by more than one Upper Layer Protocol, and that mechanisms such as CCP often apply to all connections and Protocols that flow between two peers.

If an Upper Layer Protocol specifies a method for exchanging inline threshold information, the sender can find out the receiver’s threshold value only subsequent to establishing an RPC-over-RDMA connection. The new threshold value can take effect only when a new connection is established.

7. RPC Bind Parameters

In setting up a new RDMA connection, the first action by a requester is to obtain a transport address for the responder. The mechanism used to obtain this address, and to open an RDMA connection is dependent on the type of RDMA transport, and is the responsibility of each RPC protocol binding and its local implementation.

RPC services normally register with a portmap or rpcbind [RFC1833] service, which associates an RPC program number with a service address. (In the case of UDP or TCP, the service address for NFS is normally port 2049.) This policy is no different with RDMA transports, although it may require the allocation of port numbers appropriate to each Upper Layer Protocol that uses the RPC framing defined here.

When mapped atop the iWARP transport [RFC5040] [RFC5041], which uses IP port addressing due to its layering on TCP and/or SCTP, port mapping is trivial and consists merely of issuing the port in the connection process. The NFS/RDMA protocol service address has been assigned port 20049 by IANA, for both iWARP/TCP and iWARP/SCTP.

When mapped atop InfiniBand [IB], which uses a Group Identifier (GID)-based service endpoint naming scheme, a translation MUST be employed. One such translation is defined in the InfiniBand Port Addressing Annex [IBPORT], which is appropriate for translating IP port addressing to the InfiniBand network. Therefore, in this case, IP port addressing may be readily employed by the upper layer.

When a mapping standard or convention exists for IP ports on an RDMA interconnect, there are several possibilities for each upper layer to consider:

- One possibility is to have responder register its mapped IP port with the rpcbind service, under the netid (or netid's) defined here. An RPC-over-RDMA-aware requester can then resolve its desired service to a mappable port, and proceed to connect. This is the most flexible and compatible approach, for those upper layers that are defined to use the rpcbind service.

- A second possibility is to have the responder's portmapper register itself on the RDMA interconnect at a "well known" service address. (On UDP or TCP, this corresponds to port 111.) A requester could connect to this service address and use the portmap protocol to obtain a service address in response to a program number, e.g., an iWARP port number, or an InfiniBand GID.

- Alternatively, the requester could simply connect to the mapped well-known port for the service itself, if it is appropriately defined. By convention, the NFS/RDMA service, when operating atop such an InfiniBand fabric, will use the same 20049 assignment as for iWARP.

Historically, different RPC protocols have taken different approaches to their port assignment; therefore, the specific method is left to each RPC-over-RDMA-enabled Upper Layer binding, and not addressed here.

In Section 12, "IANA Considerations", this specification defines two new "netid" values, to be used for registration of upper layers atop iWARP [RFC5040] [RFC5041] and (when a suitable port translation service is available) InfiniBand [IB]. Additional RDMA-capable networks MAY define their own netids, or if they provide a port translation, MAY share the one defined here.

8. Bi-directional RPC-over-RDMA

8.1. RPC Direction

8.1.1. Forward Direction

A traditional ONC RPC client is always a requester. A traditional ONC RPC service is always a responder. This traditional form of ONC RPC message passing is referred to as operation in the "forward direction."

During forward direction operation, the ONC RPC client is responsible for establishing transport connections.

8.1.2. Backward Direction

The ONC RPC standard does not forbid passing messages in the other direction. An ONC RPC service endpoint can act as a requester, in which case an ONC RPC client endpoint acts as a responder. This form of message passing is referred to as operation in the "backward direction."

During backward direction operation, the ONC RPC client is responsible for establishing transport connections, even though ONC RPC Calls come from the ONC RPC server.

8.1.3. Bi-direction

A pair of endpoints may choose to use only forward or only backward direction operations on a particular transport. Or, the endpoints may send operations in both directions concurrently on the same transport.

Bi-directional operation occurs when both transport endpoints act as a requester and a responder at the same time. As above, the ONC RPC client is responsible for establishing transport connections.

8.1.4. XIDs with Bi-direction

During bi-directional operation, the forward and backward directions use independent xid spaces.

In other words, a forward direction requester MAY use the same xid value at the same time as a backward direction requester on the same transport connection, but such concurrent requests use represent distinct ONC RPC transactions.

8.2. Backward Direction Flow Control

8.2.1. Backward RPC-over-RDMA Credits

Credits work the same way in the backward direction as they do in the forward direction. However, forward direction credits and backward direction credits are accounted separately.

In other words, the forward direction credit value is the same whether or not there are backward direction resources associated with an RPC-over-RDMA transport connection. The backward direction credit value MAY be different than the forward direction credit value. The rdma_credit field in a backward direction RPC-over-RDMA message MUST NOT contain the value zero.

A backward direction requester (an RPC-over-RDMA service endpoint) requests credits from the Responder (an RPC-over-RDMA client endpoint). The Responder reports how many credits it can grant. This is the number of backward direction Calls the Responder is prepared to handle at once.

When an RPC-over-RDMA server endpoint is operating correctly, it sends no more outstanding requests at a time than the client endpoint’s advertised backward direction credit value.

8.2.2. Receive Buffer Management

An RPC-over-RDMA transport endpoint must pre-post receive buffers before it can receive and process incoming RPC-over-RDMA messages. If a sender transmits a message for a receiver which has no posted receive buffer, the RDMA provider is allowed to drop the RDMA connection.

8.2.2.1. Client Receive Buffers

Typically an RPC-over-RDMA caller posts only as many receive buffers as there are outstanding RPC Calls. A client endpoint without backward direction support might therefore at times have no pre-posted receive buffers.

To receive incoming backward direction Calls, an RPC-over-RDMA client endpoint must pre-post enough additional receive buffers to match its advertised backward direction credit value. Each outstanding forward direction RPC requires an additional receive buffer above this minimum.

When an RDMA transport connection is lost, all active receive buffers are flushed and are no longer available to receive incoming messages. When a fresh transport connection is established, a client endpoint must re-post a receive buffer to handle the Reply for each retransmitted forward direction Call, and a full set of receive buffers to handle backward direction Calls.

8.2.2.2. Server Receive Buffers

A forward direction RPC-over-RDMA service endpoint posts as many receive buffers as it expects incoming forward direction Calls. That is, it posts no fewer buffers than the number of RPC-over-RDMA credits it advertises in the rdma_credit field of forward direction RPC replies.

To receive incoming backward direction replies, an RPC-over-RDMA server endpoint must pre-post a receive buffer for each backward direction Call it sends.

When the existing transport connection is lost, all active receive buffers are flushed and are no longer available to receive incoming messages. When a fresh transport connection is established, a server endpoint must re-post a receive buffer to handle the Reply for each retransmitted backward direction Call, and a full set of receive buffers for receiving forward direction Calls.

8.3. Conventions For Backward Operation

8.3.1. In the Absense of Backward Direction Support

An RPC-over-RDMA transport endpoint might not support backward direction operation. There might be no mechanism in the transport implementation to do so, or the Upper Layer Protocol consumer might not yet have configured the transport to handle backward direction traffic.

A loss of the RDMA connection may result if the receiver is not prepared to receive an incoming message. Thus a denial-of-service could result if a sender continues to send backchannel messages after every transport reconnect to an endpoint that is not prepared to receive them.

For RPC-over-RDMA Version One transports, the Upper Layer Protocol is responsible for informing its peer when it has established a backward direction capability. Otherwise even a simple backward direction NULL probe from a peer would result in a lost connection.

An Upper Layer Protocol consumer MUST NOT perform backward direction ONC RPC operations unless the peer consumer has indicated it is prepared to handle them. A description of Upper Layer Protocol mechanisms used for this indication is outside the scope of this document.

8.3.2. Backward Direction Retransmission

In rare cases, an ONC RPC transaction cannot be completed within a certain time. This can be because the transport connection was lost, the Call or Reply message was dropped, or because the Upper Layer consumer delayed or dropped the ONC RPC request. Typically, the requester sends the transaction again, reusing the same RPC XID. This is known as an "RPC retransmission".

In the forward direction, the Caller is the ONC RPC client. The client is always responsible for establishing a transport connection before sending again.

In the backward direction, the Caller is the ONC RPC server. Because an ONC RPC server does not establish transport connections with clients, it cannot send a retransmission if there is no transport connection. It must wait for the ONC RPC client to re-establish the transport connection before it can retransmit ONC RPC transactions in the backward direction.

If an ONC RPC client has no work to do, it may be some time before it re-establishes a transport connection. Backward direction Callers must be prepared to wait indefinitely before a connection is established before a pending backward direction ONC RPC Call can be retransmitted.

8.3.3. Backward Direction Message Size

RPC-over-RDMA backward direction messages are transmitted and received using the same buffers as messages in the forward direction. Therefore they are constrained to be no larger than receive buffers posted for forward messages. Typical implementations have chosen to use 1024-byte buffers.

It is expected that the Upper Layer Protocol consumer establishes an appropriate payload size limit for backward direction operations, either by advertising that size limit to its peers, or by convention. If that is done, backward direction messages do not exceed the size of receive buffers at either endpoint.

If a sender transmits a backward direction message that is larger than the receiver is prepared for, the RDMA provider drops the message and the RDMA connection.

If a sender transmits an RDMA message that is too small to convey a complete and valid RPC-over-RDMA and RPC message in either direction, the receiver MUST NOT use any value in the fields that were transmitted. Namely, the rdma_credit field MUST be ignored, and the message dropped.

8.3.4. Sending A Backward Direction Call

To form a backward direction RPC-over-RDMA Call message on an RPC-over-RDMA Version One transport, an ONC RPC service endpoint constructs an RPC-over-RDMA header containing a fresh RPC XID in the rdma_xid field.

The rdma_vers field MUST contain the value one. The number of requested credits is placed in the rdma_credit field.

The rdma_proc field in the RPC-over-RDMA header MUST contain the value RDMA_MSG. All three chunk lists MUST be empty.

The ONC RPC Call header MUST follow immediately, starting with the same XID value that is present in the RPC-over-RDMA header. The Call header’s msg_type field MUST contain the value CALL.

8.3.5. Sending A Backward Direction Reply

To form a backward direction RPC-over-RDMA Reply message on an RPC-over-RDMA Version One transport, an ONC RPC client endpoint constructs an RPC-over-RDMA header containing a copy of the matching ONC RPC Call’s RPC XID in the rdma_xid field.

The rdma_vers field MUST contain the value one. The number of granted credits is placed in the rdma_credit field.

The rdma_proc field in the RPC-over-RDMA header MUST contain the value RDMA_MSG. All three chunk lists MUST be empty.

The ONC RPC Reply header MUST follow immediately, starting with the same XID value that is present in the RPC-over-RDMA header. The Reply header’s msg_type field MUST contain the value REPLY.

8.4. Backward Direction Upper Layer Binding

RPC programs that operate on RPC-over-RDMA Version One only in the backward direction do not require an Upper Layer Binding specification. Because RPC-over-RDMA Version One operation in the backward direction does not allow chunking, there can be no DDP-eligible data items in such a program. Backward direction operation occurs on an already-established connection, thus there is no need to specify RPC bind parameters.

9. Transport Protocol Extensibility

RPC programs are defined solely by their XDR definitions. They are independent of the transport mechanism used to convey base RPC messages. Protocols defined by XDR often have signifcant extensibility restrictions placed on them.

Not all extensibility restrictions on RPC-based Upper Layer Protocols may be appropriate for an RPC transport protocol. TCP [RFC0793], for example, is an RPC transport protocol that has been extended many times independently of the RPC and XDR standards.

RPC-over-RDMA might be considered as an extension of the RPC protocol rather than a separate transport, however.

- The mechanisms that TCP uses to move data are opaque to the RPC implementation and RPC programs using it. Upper Layer Protocols are often aware that RPC-over-RDMA is present, as they identify data items that can be moved via direct data placement.

- RPC-over-RDMA is used only for moving RPC messages, and not ever for generic data transfer.

- RPC-over-RDMA relies on a more sophisticated set of base transport operations than traditional socket-based transports. Interoperability depends on RPC-over-RDMA implementations using these operations in a predictable way.

- The RPC-over-RDMA header is specified using XDR, unlike other RPC transport protocols.

9.1. Bumping The RPC-over-RDMA Version

Place holder section.

Because the version number is encoded as part of the RPC-over-RDMA header and the RDMA_ERROR message type is used to indicate errors, these first four fields and the start of the chunk lists MUST always remain aligned at the same fixed offsets for all versions of the RPC-over-RDMA header.

The value of the RPC-over-RDMA header’s version field MUST be changed

- Whenever the on-the-wire format of the RPC-over-RDMA header is changed in a way that prevents interoperability with current implementations

- Whenever the set of abstract RDMA operations that may be used is changed

- Whenever the set of allowable transfer models is altered

10. Security Considerations

A primary consideration is the protection of the integrity and privacy of local memory by the RDMA transport itself. The use of RPC-over-RDMA MUST NOT introduce any vulnerabilities to system memory contents, or to memory owned by user processes. These protections are provided by the RDMA layer specifications, and specifically their security models.

It is REQUIRED that any RDMA provider used for RPC transport be conformant to the requirements of [RFC5042] in order to satisfy these protections. Best practices to ensure memory contents are completely protected during an RPC transaction include the following.

- The use of Protection Domains to limit the exposure of memory regions to a single connection is critical. Any attempt by a host not participating in that connection to re-use handles will result in a connection failure. Because Upper Layer Protocol security mechanisms rely on this aspect of Reliable Connection behavior, strong authentication of the remote is recommended.

- Unpredictable memory handles should be used for any operation requiring advertised memory regions. Advertising a continuously registered memory region allows a remote host to read or write to that region even when an RPC involving that memory is not under way. Therefore advertising persistently registered memory should be avoided.

- Advertised memory regions should be invalidated as soon as related RPC operations are complete. Invalidation and DMA unmapping of regions should be complete before an RPC application is allowed to continue execution and use the contents of a memory region.

Once delivered securely by the RDMA provider, any RDMA-exposed addresses will contain only RPC payloads in the chunk lists, transferred under the protection of RPCSEC_GSS integrity and privacy. By these means, the data will be protected end-to-end, as required by the RPC layer security model.

RPC provides its own security via the RPCSEC_GSS framework [RFC2203]. RPCSEC_GSS can provide message authentication, integrity checking, and privacy. This security mechanism is unaffected by the RDMA transport. However, there is much data movement associated with computation and verification of integrity, or encryption/decryption, so certain performance advantages may be lost.

For efficiency, a more appropriate security mechanism for RDMA links may be link-level protection, such as certain configurations of IPsec, which may be co-located in the RDMA hardware. The use of link-level protection MAY be negotiated through the use of the RPCSEC_GSS mechanism defined in [RFC5403] in conjunction with the Channel Binding mechanism [RFC5056] and IPsec Channel Connection Latching [RFC5660]. Use of such mechanisms is REQUIRED where integrity and/or privacy is desired, and where efficiency is required.

11. IANA Considerations

Three new assignments are specified by this document:

- A new set of RPC "netids" for resolving RPC-over-RDMA services

- Optional service port assignments for Upper Layer Bindings

- An RPC program number assignment for the configuration protocol

These assignments have been established, as below.

The new RPC transport has been assigned an RPC "netid", which is an rpcbind [RFC1833] string used to describe the underlying protocol in order for RPC to select the appropriate transport framing, as well as the format of the service addresses and ports.

The following "Netid" registry strings are defined for this purpose:

```
NC_RDMA "rdma"
NC_RDMA6 "rdma6"
```

These netids MAY be used for any RDMA network satisfying the requirements of Section 2, and able to identify service endpoints using IP port addressing, possibly through use of a translation service as described above in Section 10, "RPC Binding". The "rdma" netid is to be used when IPv4 addressing is employed by the underlying transport, and "rdma6" for IPv6 addressing.

The netid assignment policy and registry are defined in [RFC5665].

As a new RPC transport, this protocol has no effect on RPC program numbers or existing registered port numbers. However, new port numbers MAY be registered for use by RPC-over-RDMA-enabled services, as appropriate to the new networks over which the services will operate.

For example, the NFS/RDMA service defined in [RFC5667] has been assigned the port 20049, in the IANA registry:

```
nfsrdma 20049/tcp Network File System (NFS) over RDMA
nfsrdma 20049/udp Network File System (NFS) over RDMA
nfsrdma 20049/sctp Network File System (NFS) over RDMA
```

The RPC program number assignment policy and registry are defined in [RFC5531].

12. Acknowledgments

The editor gratefully acknowledges the work of Brent Callaghan and Tom Talpey on the original RPC-over-RDMA Version One specification [RFC5666].

The comments and contributions of Karen Deitke, William Simpson, Dai Ngo, Chunli Zhang, Dominique Martinet, and Mahesh Siddheshwar are accepted with many and great thanks. The editor also wishes to thank Dave Noveck and Bill Baker for their unwavering support of this work.

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

13. Appendices

13.1. Appendix 1: XDR Examples

RPC-over-RDMA chunk lists are complex data types. In this appendix, illustrations are provided to help readers grasp how chunk lists are represented inside an RPC-over-RDMA header.

An RDMA segment is the simplest component, being made up of a 32-bit handle (H), a 32-bit length (L), and 64-bits of offset (OO). Once flattened into an XDR stream, RDMA segments appear as

```
HLOO
```

A Read segment has an additional 32-bit position field. Read segments appear as

```
PHLOO
```

A Read chunk is a list of Read segments. Each segment is preceded by a 32-bit word containing a one if there is a segment, or a zero if there are no more segments (optional-data). In XDR form, this would look like

```
1 PHLOO 1 PHLOO 1 PHLOO 0
```

where P would hold the same value for each segment belonging to the same Read chunk.

The Read List is also a list of Read segments. In XDR form, this would look a lot like a Read chunk, except that the P values could vary across the list. An empty Read List is encoded as a single 32-bit zero.

One Write chunk is a counted array of segments. In XDR form, the count would appear as the first 32-bit word, followed by an HLOO for each element of the array. For instance, a Write chunk with three elements would look like

```
   3 HLOO HLOO HLOO
```

The Write List is a list of counted arrays. In XDR form, this is a combination of optional-data and counted arrays. To represent a Write List containing a Write chunk with three segments and a Write chunk with two segments, XDR would encode

```
   1 3 HLOO HLOO HLOO 1 2 HLOO HLOO 0
```

An empty Write List is encoded as a single 32-bit zero.

The Reply chunk is the same as a Write chunk. Since it is an optional-data field, however, there is a 32-bit field in front of it that contains a one if the Reply chunk is present, or a zero if it is not. After encoding, a Reply chunk with 2 segments would look like

```
   1 2 HLOO HLOO
```

Frequently a requester does not provide any chunks. In that case, after the four fixed fields in the RPC-over-RDMA header, there are simply three 32-bit fields that contain zero.

14. References

14.1. Normative References

[RFC1833] Srinivasan, R., "Binding Protocols for ONC RPC Version 2", RFC 1833, DOI 10.17487/RFC1833, August 1995, <http://www.rfc-editor.org/info/rfc1833>.

[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>.

[RFC2203] Eisler, M., Chiu, A., and L. Ling, "RPCSEC_GSS Protocol Specification", RFC 2203, DOI 10.17487/RFC2203, September 1997, <http://www.rfc-editor.org/info/rfc2203>.

[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>.

[RFC5042] Pinkerton, J. and E. Deleganes, "Direct Data Placement Protocol (DDP) / Remote Direct Memory Access Protocol (RDMAP) Security", RFC 5042, DOI 10.17487/RFC5042, October 2007, <http://www.rfc-editor.org/info/rfc5042>.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, DOI 10.17487/RFC5056, November 2007, <http://www.rfc-editor.org/info/rfc5056>.

[RFC5403] Eisler, M., "RPCSEC_GSS Version 2", RFC 5403, DOI 10.17487/RFC5403, February 2009, <http://www.rfc-editor.org/info/rfc5403>.

[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>.

[RFC5660] Williams, N., "IPsec Channels: Connection Latching", RFC 5660, DOI 10.17487/RFC5660, October 2009, <http://www.rfc-editor.org/info/rfc5660>.

[RFC5665] Eisler, M., "IANA Considerations for Remote Procedure Call (RPC) Network Identifiers and Universal Address Formats", RFC 5665, DOI 10.17487/RFC5665, January 2010, <http://www.rfc-editor.org/info/rfc5665>.

[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>.

14.2. Informative References

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

[IBPORT] InfiniBand Trade Association, "IP Addressing Annex", <http://www.infinibandta.org>.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

[RFC1094] Nowicki, B., "NFS: Network File System Protocol specification", RFC 1094, DOI 10.17487/RFC1094, March 1989, <http://www.rfc-editor.org/info/rfc1094>.

[RFC1813] Callaghan, B., Pawlowski, B., and P. Staubach, "NFS Version 3 Protocol Specification", RFC 1813, DOI 10.17487/RFC1813, June 1995, <http://www.rfc-editor.org/info/rfc1813>.

[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>.

[RFC5532] Talpey, T. and C. Juszczak, "Network File System (NFS) Remote Direct Memory Access (RDMA) Problem Statement", RFC 5532, DOI 10.17487/RFC5532, May 2009, <http://www.rfc-editor.org/info/rfc5532>.

[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>.

[RFC5667] Talpey, T. and B. Callaghan, "Network File System (NFS) Direct Data Placement", RFC 5667, DOI 10.17487/RFC5667, January 2010, <http://www.rfc-editor.org/info/rfc5667>.

[RFC7530] Haynes, T., Ed. and D. Noveck, Ed., "Network File System (NFS) Version 4 Protocol", RFC 7530, DOI 10.17487/RFC7530, March 2015, <http://www.rfc-editor.org/info/rfc7530>.

Authors' Addresses

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

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

Tom Talpey
Microsoft Corp.
One Microsoft Way
Redmond, WA 98052
USA

Phone: +1 425 704-9945
Email: ttalpey@microsoft.com