Network File System Version 4
Internet-Draft
Obsoletes: 5666 (if approved)
Intended status: Standards Track
Expires: July 14, 2016

C. Lever, Ed.
Oracle
W. Simpson
DayDreamer
T. Talpey
Microsoft
January 11, 2016

Remote Direct Memory Access Transport for Remote Procedure Call
draft-ietf-nfsv4-rfc5666bis-02

Abstract

This document specifies a protocol for conveying Remote Procedure Call (RPC) messages on physical transports capable of Remote Direct Memory Access (RDMA). It requires no revision to application RPC protocols 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; however, the protocol specified by this document is based on existing interoperating implementations of the RPC-over-RDMA Version One protocol. The new specification clarifies text that is subject to multiple interpretations and removes support for unimplemented RPC-over-RDMA Version One protocol elements. This document makes the role of Upper Layer Bindings an explicit part of the specification. In addition, this document introduces conventions that enable bi-directional RPC-over-RDMA operation to allow operation of NFSv4.1 [RFC5661] on RDMA transports.

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 On RDMA Transports

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

Open Network Computing Remote Procedure Call (ONC RPC, or simply, RPC) [RFC5531] is a remote procedure call protocol that runs over a variety of transports. Most RPC implementations today use UDP or TCP. 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 specific fashion that must be fully described if RPC implementations are to interoperate.

RDMA transports present semantics different from either UDP or TCP. They retain message delineations like UDP, but provide a reliable and sequenced data transfer like TCP. They also provide an offloaded bulk transfer service not provided by UDP or TCP. RDMA transports are therefore appropriately viewed as a new transport type by RPC.

In this context, the Network File System (NFS) protocols as described in [RFC1094], [RFC1813], [RFC7530], [RFC5661], and future NFSv4 minor verions are obvious beneficiaries of RDMA transports. A complete problem statement is discussed in [RFC5532], and NFSv4-related issues are discussed in [RFC5661]. Many other RPC-based protocols can also benefit.

Although the RDMA transport described here can provide relatively transparent support for any RPC application, this document also describes mechanisms that can optimize data transfer further, given more active participation by RPC applications.

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. The section numbers below refer to [RFC5666].

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

- Section 3 has been re-organized and split into sub-sections to help implementers locate specific requirements and definitions.

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

- 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.

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

- A section outlining requirements for Upper Layer Bindings has been added.

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

2.2. Changes To The Protocol

While the protocol described herein interoperates with existing implementations of [RFC5666], the following changes have been made relative to the protocol described in that document:

- Support for the Read-Read transfer model has been removed. Read-Read is a slower transfer model than Read-Write, thus implementers have chosen not to support it. This simplifies explanatory text, and support for the RDMA_DONE message type is no longer necessary.

- The specification of RDMA_MSGP in [RFC5666] and current implementations of it are incomplete. Therefore the RDMA_MSGP message type is no longer supported.

- Technical errors with regard to handling RPC-over-RDMA header errors have been corrected.

- Specific requirements related to handling XDR round-up and abstract data types have been added. Responders are now forbidden from writing Write chunk round-up bytes.

- 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 [RFC5661] backchannel on RPC-over-RDMA Version One transports when both endpoints support the new functionality.

The protocol version number has not been 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. The term Upper Layer Protocol refers to an RPC Program and Version tuple, which is a versioned set of procedure calls that comprise a single well-defined API. One example of an Upper Layer 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 that work be done. 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. At this point the RPC transaction designated by the xid in the call message is terminated and the xid is retired.

3.1.3. RPC Transports

The role of an "RPC transport" is to mediate the exchange of RPC messages between requesters and responders. An RPC transport bridges the gap between the RPC message abstraction and the native operations of a particular network transport.

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.

3.1.4. External Data Representation

In a heterogenous environment, one cannot assume that requesters and responders represent data the same way. RPC uses eXternal Data Representation, or XDR, to translate data types and serialize arguments and results [RFC4506].

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 complex data types so they can be conveyed as a 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.

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

3.1.4.2. 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 as it is encoded 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, and the zero bytes are not exposed to the Upper Layer.

This technique is referred to as "XDR round-up," and the extra octets are referred to as "XDR padding".

3.2. Remote Direct Memory Access

RPC requesters and responders 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 enables both optimizations.

3.2.1. Direct Data Placement

Typically, RPC implementations copy the contents of RPC messages into a buffer before being sent. An efficient RPC implementation sends 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. Although this may not be efficient, before an RDMA transfer a sender may copy data into an intermediate buffer before an RDMA transfer. After an RDMA transfer, a 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 to another location 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].

Registered Memory
   Registered memory is a segment of memory that is assigned a steering tag that temporarily permits access by the RDMA provider

to perform data transfer operations. The RPC-over-RDMA Version One 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 Send
: The RDMA provider supports an RDMA Send operation, with completion signaled on the receiving peer after data has been placed in a pre-posted memory segment. Sends complete at the receiver in the order they were issued at the sender. The amount of data transferred by an RDMA Send operation is limited by the size of the remote pre-posted memory segment.

RDMA Receive
: The RDMA provider supports an RDMA Receive operation to receive data conveyed by incoming RDMA Send operations. To reduce the amount of memory that must remain pinned awaiting incoming Sends, the amount of pre-posted memory is limited. Flow-control to prevent overrunning receiver resources is provided by the RDMA consumer (in this case, the RPC-over-RDMA Version One protocol).

RDMA Write
: The RDMA provider supports an RDMA Write operation to directly place data in remote memory. The local host initiates an RDMA Write, and completion is signaled there. No completion is signaled on the remote. The local host provides a steering tag, memory address, and length of the remote's memory segment.

  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 write initiator guarantees that prior RDMA Write data has been successfully placed in the remote peer's memory.

RDMA Read
: The RDMA provider supports an RDMA Read operation to directly place peer source data in the read initiator's memory. The local host initiates an RDMA Read, and completion is signaled there; no completion is signaled on the remote. The local host provides steering tags, memory addresses, and a length for the remote source and local destination memory segments.

  The remote peer receives no notification of RDMA Read completion. The local host signals completion as part of an RDMA Send message so that the remote peer can release steering tags and subsequently free associated source memory segments.

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].

4. RPC-Over-RDMA 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 pull RPC arguments or whole RPC calls from the requester. Requesters employ RDMA Read operations to pull RPC results or whole RPC relies from the responder.

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

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

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

[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 no longer supported. Other Transfer Models may be used by a future version of RPC-over-RDMA.

4.2. RPC Message Framing

On an RPC-over-RDMA transport, each RPC message is encapsulated by an RPC-over-RDMA message. An RPC-over-RDMA message consists of two XDR streams.

Transport-Specific Stream
: The "transport-specific XDR stream," or "Transport stream," contains an RPC-over-RDMA header that describes and controls the transfer of the Payload stream in this RPC-over-RDMA message. 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.

RPC Payload XDR Stream
: The "RPC payload stream," or "Payload stream", contains the encapsulated RPC message being transferred by this RPC-over-RDMA message.

In its simplest form, an RPC-over-RDMA message consists of a Transport stream followed immediately by a Payload stream conveyed together in a single RDMA Send. To transmit large RPC messages, a combination of one RDMA Send operation and one or more RDMA Read or Write operations is employed.

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 underlying 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.

Flow control for RDMA Send operations directed to the responder is implemented as a simple request/grant protocol in the RPC-over-RDMA header associated with each RPC message (Section 5.2.3 has details).

- o The RPC-over-RDMA header for RPC call messages contains a requested credit value for the responder. This is the maximum number of RPC replies the requester can handle at once, independent of how many RPCs are in flight at that moment. The requester MAY dynamically adjust the requested credit value to match its expected needs.

- o The RPC-over-RDMA header for RPC reply messages provides the granted result. This is the maximum number of RPC calls the responder can handle at once, without regard to how many RPCs are in flight at that moment. The granted value MUST NOT be zero, since such a value would result in deadlock. The responder MAY dynamically adjust the granted credit value to match its needs or policies (e.g. to accommodate the available resources in a shared receive queue).

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 RDMA layer error does not occur, the responder MAY handle excess requests or return an RPC layer error to the requester.

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 lower of the requested and granted credit values, minus the number of requests in flight. Advertised credit values are not altered when individual RPCs are started or completed.

On occasion a requester or responder may need to adjust the amount of resources available to a connection. When this happens, the responder needs to ensure that a credit increase is effected (i.e. receives are posted) before the next reply is sent.

Certain RDMA implementations may impose additional flow control restrictions, such as limits on RDMA Read operations in progress at the responder. Accommodation of such restrictions is considered the responsibility of each RPC-over-RDMA Version One implementation.

4.3.1. Initial Connection State

There are two operational parameters for each connection:

Credit Limit

As described above, the total number of responder receive buffers is sometimes referred to as 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
A receiver's "inline threshold" value is the largest message size (in bytes) that can be conveyed via an RDMA Send/Receive combination. Each connection has two inline threshold values, one for each peer receiver.

Unlike the connection's credit limit, inline threshold values are 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.

As a basis for an initial exchange of RPC requests, each RPC-over-RDMA Version One connection provides 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.

Receiver implementations MUST support an inline threshold of 1024 bytes, but MAY support larger inline thresholds values. A mechanism for discovering a peer's inline threshold value before a connection is established may be used to optimize Send operations. In the absense of such a mechanism, senders MUST assume a receiver's inline threshold is 1024 bytes.

4.4. XDR Encoding With Chunks

XDR data items in an RPC message are encoded as a contiguous sequence of bytes for network transmission. This sequence of bytes is known as an XDR stream. In the case of an RDMA transport, during XDR encoding it can be determined that an XDR data item is large enough that it might be more efficient if the transport placed the content of the data item directly in the receiver's memory.

4.4.1. Reducing An XDR Stream

RPC-over-RDMA Version One provides a mechanism for moving part of an RPC message via a data transfer separate from an RDMA Send/Receive. The sender removes one or more XDR data items from the Payload stream. They are conveyed via one or more RDMA Read or Write operations. The receiver inserts the data items into the Payload stream before passing it to the Upper Layer.

A contiguous piece of a Payload stream that is split out and moved via separate RDMA operations is known as a "chunk." A Payload stream after chunks have been removed is referred to as a "reduced" Payload stream.

4.4.2. DDP-Eligibility

Only an XDR data item that might benefit from Direct Data Placement may be reduced. The eligibility of particular XDR data items to be reduced is not specified by this document.

To maintain interoperability on an RPC-over-RDMA transport, a determination of which XDR data items in each Upper Layer Protocol 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 an Upper Layer Protocol to use an RPC-over-RDMA transport is known as an "Upper Layer Binding specification," or ULB.

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

Detailed requirements for Upper Layer Bindings are discussed in full in Section 8.

4.4.3. RDMA Segments

When encoding a Payload stream that contains a DDP-eligible data item, a sender may choose to reduce that data item. It does not place the item into the Payload stream. Instead, the sender records in the RPC-over-RDMA header the actual address and size of the memory region containing that data item.

The requester provides location information for DDP-eligible data items in both RPC calls and replies. The responder uses this information to initiate RDMA Read and Write operations to retrieve or update the content of the requester's memory.

An "RDMA segment", or just "segment", is an RPC-over-RDMA header data object that contains 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 each 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.

4.4.4. Chunks

In RPC-over-RDMA Version One, a "chunk" refers to a portion of the Payload stream that is moved via RDMA Read or Write operations. Chunk data is removed from the sender's Payload stream, transferred by separate RDMA operations, and then re-inserted into the receiver's Payload 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. Not every message has chunks associated with it.

4.4.4.1. Counted Arrays

If a chunk contains a counted array data type, the count of array elements MUST remain in the Payload 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 Payload stream, while the bytes in the array are removed from the Payload stream and transferred within the chunk.

Any byte count left in the Payload 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 a 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 Payload stream, but each enclosed array, including its item count, is transferred as part of the chunk.

4.4.4.2. Optional-data

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

4.4.4.3. XDR Unions

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

4.4.5. 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 RDMA segment in a Read chunk has an additional Position field.

Position
: The byte offset in the Payload stream where the receiver re-inserts the data item conveyed in a chunk. The Position value MUST be computed from the beginning of the Payload stream, which begins at Position zero. All segments belonging to the same Read chunk have the same value in their Position field.

While constructing an RPC-over-RDMA Call message, a requester registers memory segments containing data in Read chunks. It advertises these chunks in the RPC-over-RDMA header of the RPC call.

After receiving an RPC call sent via an RDMA Send operation, a 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 received Payload stream at the Position value recorded in the segment.

Put another way, a receiver inserts the first segment in a Read chunk into the Payload stream at the byte offset indicated by its Position field. Segments whose Position field value match this offset are concatenated afterwards, until there are no more segments at that Position value. The next XDR data item in the Payload stream follows.

4.4.5.1. 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 in the XDR stream can start on a four-byte boundary. Receivers ignore the content of the pad bytes.

After an XDR data item has been reduced, all data items remaining in the Payload stream must continue to adhere to these padding requirements. Thus when an XDR data item is moved from the Payload stream into a Read chunk, the requester MUST remove XDR padding for that data item from the Payload stream as well.

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 the reconstructed call 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.

4.4.5.2. Decoding Read Chunks

When decoding an RPC-over-RDMA message, the responder first decodes the chunk lists from the RPC-over-RDMA header, then proceeds to decode the Payload stream. Whenever the XDR offset in the Payload 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.

The responder acknowledges its completion of use of Read chunk source buffers when it replies to the requester. The requester may then release Read chunks advertised in the request.

4.4.6. Write Chunks

A "Write chunk" represents an XDR data item that is to be pushed from a responder to a 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 has prepared the reply Payload stream.

While constructing an RPC call message, a requester also prepares memory regions to catch DDP-eligible reply data items. A requester does not know the actual length of the result data item to be returned, thus it MUST register a Write chunk long enough to accommodate the maximum possible size of the returned data item.

A responder copies the requester-provided Write chunk segments into the RPC-over-RDMA header that it returns with the reply. The responder updates the segment length fields to reflect the actual amount of data that is being returned in the Write 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.

4.4.6.1. 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 define 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.

4.4.6.2. 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 in the XDR stream can start on a four-byte boundary. Receivers ignore the content of the pad bytes.

After a data item is reduced, data items remaining in the Payload stream must continue to adhere to these padding requirements. Thus when an XDR data item is moved from a reply Payload stream into a Write chunk, the responder MUST remove XDR padding for that data item from the reply Payload stream as well.

A requester SHOULD NOT provide extra length in a Write chunk to accommodate XDR pad bytes. A responder MUST NOT write XDR pad bytes for a Write chunk.

4.5. Message Size

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

4.5.1. Short Messages

RPC messages are frequently smaller than typical inline thresholds. 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 receiver's inline threshold is to append the Payload stream directly to the Transport stream. 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.

4.5.2. Chunked Messages

If DDP-eligible data items are present in a Payload stream, a sender MAY reduce the Payload stream and use RDMA Read or Write operations to move the reduced data items. The Transport stream with the reduced Payload stream immediately following is transferred using a single RDMA Send operation.

After receiving the Transport and Payload streams of a Chunked RPC-over-RDMA Call message, the responder uses RDMA Read operations to move reduced data items in Read chunks. Before sending the Transport and Payload streams of a Chunked RPC-over-RDMA Reply message, the responder uses RDMA Write operations to move reduced data items in Write and Reply chunks.

4.5.3. Long Messages

When a Payload stream is larger than the receiver's inline threshold, the Payload stream is reduced by removing DDP-eligible data items and placing them in chunks to be moved separately. If there are no DDP-eligible data items in the Payload stream, or the Payload stream is still too large after it has been reduced, the RDMA transport MUST use RDMA Read or Write operations to convey the Payload stream itself. This mechanism is referred to as a "Long Message."

To transmit a Long Message, the sender conveys only the Transport stream with an RDMA Send operation. The Payload stream is not included in the Send buffer in this instance. Instead, the requester provides chunks that the responder uses to move the Payload stream.

Long RPC call
: To send a Long RPC-over-RDMA Call message, the requester provides a special Read chunk that contains the RPC call's Payload stream. Every segment in this Read chunk MUST contain zero in its Position field. Thus this chunk is known as a "Position Zero Read chunk."

Long RPC reply
: To send a Long RPC-over-RDMA Reply message, the requester provides a single special Write chunk in advance, known as the "Reply chunk", that will contain the RPC reply's Payload stream. The requester sizes the Reply chunk to accommodate the maximum expected reply size 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 MUST send a Long reply whenever a Reply chunk has been provided by a requester.

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

Every RPC-over-RDMA Version One message has a header that includes a copy of the message's transaction ID, data for managing RDMA flow control credits, and lists of RDMA segments used for RDMA Read and

Write operations. All RPC-over-RDMA header content is contained in the Transport stream, and thus MUST be XDR encoded.

RPC message layout is unchanged from that described in [RFC5531] except for the possible reduction of data items that are moved by RDMA Read or Write operations.

5.1. XDR Protocol Definition

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_header {
              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 rdma_err) {
              case RDMA_ERR_VERS:
                uint32 rdma_vers_low;
                uint32 rdma_vers_high;
              case RDMA_ERR_CHUNK:
                void;
      };

      union rdma_body switch (rpcrdma1_proc rdma_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>
```

5.2. 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:

5.2.1. 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 message. 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 message, if it so chooses.

5.2.2. 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.

5.2.3. 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.

5.2.4. Message type

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

- o 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.

- o RDMA_MSGP = 2 is reserved.

- o RDMA_DONE = 3 is reserved.

- o RDMA_ERROR = 4 is used to signal an error in RDMA chunk encoding.

An RDMA_MSG type message conveys the Transport stream and the Payload stream via an RDMA Send operation. The Transport stream contains the four fixed fields, followed by the Read and Write lists and the Reply chunk, though any or all three MAY be marked as not present. The Payload stream then follows, beginning with its XID field. If a Read or Write chunk list is present, a portion of the Payload stream has been excised and is conveyed separately via RDMA Read or Write operations.

An RDMA_NOMSG type message conveys the Transport stream via an RDMA Send operation. The Transport stream contains the four fixed fields, followed by the Read and Write chunk lists and the Reply chunk. Though any MAY be marked as not present, one MUST be present and MUST hold the Payload stream for this RPC-over-RDMA message, beginning with its XID field. If a Read or Write chunk list is present, a portion of the Payload stream has been excised and is conveyed separately via RDMA Read or Write operations.

An RDMA_ERROR type message conveys the Transport stream via an RDMA Send operation. The Transport stream contains the four fixed fields, followed by formatted error information. No Payload stream is conveyed in this type of RPC-over-RDMA message.

A gather operation on each RDMA Send operation can be used to marshal the Transport and Payload streams separately. However, the total length of the gathered send buffers MUST NOT exceed the peer receiver's inline threshold.

5.3. 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 5.7 as an aid to understanding.

5.3.1. Read List

Each RDMA_MSG or RDMA_NOMSG type message 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 location of data the responder is to retrieve via RDMA Read operations.

Via a Position Zero Read Chunk, a requester may provide an RPC Call message as a chunk in the Read list.

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

5.3.2. Write List

Each RDMA_MSG or RDMA_NOMSG type message 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 via RDMA Write operations.

When a Write list is provided for the results of an RPC Call, the responder MUST provide any corresponding data via RDMA Write to the memory referenced in the chunk's segments. The Write list is 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.

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.

5.3.3. Reply Chunk

Each RDMA_MSG or RDMA_NOMSG type message 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 requester MUST provide a Reply chunk whenever the maximum possible size of the reply is larger than its own inline threshold. The Reply chunk MUST be large enough to contain a Payload stream (RPC message) of this maximum size.

When a Reply chunk is provided, a responder MUST convey the RPC reply message in this chunk.

5.4. Memory Registration

RDMA requires that data is transferred between only registered memory segments at the source and destination. All protocol headers as well as separately transferred data chunks must reside in registered memory.

Since the cost of registering and de-registering memory can be a significant proportion of the RDMA transaction cost, it is important to minimize registration activity. This can be achieved within RPC-controlled memory by allocating chunk list data and RPC headers in a reusable way from pre-registered pools.

5.4.1. Registration Longevity

Data chunks transferred via RDMA Read and Write MAY reside in a memory allocation 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 endpoint must ensure that these memory segments are properly fenced from the responder before allowing Upper Layer access to the data contained in them. The data in such segments must be at rest while a responder has access to that memory.

This includes segments that are associated with canceled RPCs. A responder cannot know that the requester is no longer waiting for a reply, and might proceed to read or even update memory that the requester has released for other use.

5.4.2. Communicating DDP-Eligibility

The interface by which an Upper Layer Protocol implementation communicates the eligibility of a data item locally to its local RPC-over-RDMA endpoint is not described by this specification.

Depending on the implementation and constraints imposed by Upper Layer Bindings, it is possible to implement reduction transparently to upper layers. 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.

5.4.3. Registration Strategies

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. Error Handling

A receiver performs basic validity checks on the RPC-over-RDMA header and chunk contents before it passes the RPC message to the RPC consumer. If errors are detected in an RPC-over-RDMA header, an RDMA_ERROR type message MUST be generated. Because the transport layer may not be aware of the direction of a problematic RPC message, an RDMA_ERROR type message MAY be generated by either a requester or a responder.

To form an RDMA_ERROR type message: The rdma_xid field MUST contain the same XID that was in the rdma_xid field in the failing request; The rdma_vers field MUST contain the same version that was in the rdma_vers field in the failing request; The rdma_proc field MUST contain the value RDMA_ERROR; The rdma_err field contains a value that reflects the type of error that occurred, as described below.

An RDMA_ERROR type message indicates a permanent error. When receiving an RDMA_ERROR type message, a requester should attempt to terminate the RPC transaction if it recognizes the XID in the reply's rdma_xid field, and return an error to the application to prevent retrying the failed RPC transaction.

To avoid an infinite loop, a receiver should drop an RDMA_ERROR type message that is malformed.

5.5.1. Header Version Mismatch

When a receiver detects an RPC-over-RDMA header version that it does not support (currently this document defines only Version One), it MUST reply with an rdma_err value of ERR_VERS, providing the low and high inclusive version numbers it does, in fact, support.

5.5.2. XDR Errors

A receiver might encounter an XDR parsing error that prevents it from processing the incoming Transport stream. Examples of such errors include an invalid value in the rdma_proc field, an RDMA_NOMSG message that has no chunk lists, or the contents of the rdma_xid field might not match the contents of the XID field in the accompanying RPC message. In such cases, the responder MUST reply with an rdma_err value of ERR_CHUNK.

When a responder receives a valid RPC-over-RDMA header but the responder's Upper Layer Protocol implementation cannot parse the RPC arguments in the RPC Call message, the responder SHOULD return a RPC_GARBAGEARGS reply, using an RDMA_MSG type message. This type of parsing failure might be due to mismatches between chunk sizes or offsets and the contents of the Payload stream, for example. A responder MAY also report the presence of a non-DDP-eligible data item in a Read or Write chunk using RPC_GARBAGEARGS.

5.5.3. Responder Operational Errors

Problems can arise as a responder attempts to use requester-provided resources for RDMA Read or Write operations. For example:

- Chunks can be validated only by using their contents to form RDMA Read or Write operations. If chunk contents are invalid (say, a segment is no longer registered, or a chunk length is too long), a Remote Access error occurs.

- If a requester's receive buffer is too small, the responder's Send operation completes with a Local Length Error.

- If the requester-provided Reply chunk is too small to accommodate a large RPC reply, a Remote Access error occurs. A responder can detect this problem before attempting to write past the end of the Reply chunk.

Operational errors are typically fatal to the connection. To avoid a retransmission loop and repeated connection loss that deadlocks the connection, once the requester has re-established a connection, the responder should send an RDMA_ERROR reply with an rdma_err value of ERR_CHUNK to indicate that no RPC-level reply is possible for that XID.

5.5.4. RDMA Transport Errors

The RDMA connection and physical link 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.

5.6. Protocol Elements No Longer Supported

The following protocol elements are no longer supported in RPC-over-RDMA Version One. Related enum values and structure definitions remain in the RPC-over-RDMA Version One protocol for backwards compatibility.

5.6.1. RDMA_MSGP

The specification of RDMA_MSGP in Section 3.9 of [RFC5666] is incomplete. To fully specify RDMA_MSGP would require:

- Updating the definition of DDP-eligibility to include data items that may be transferred, with padding, via RDMA_MSGP type messages

- Adding full operational descriptions of the alignment and threshold fields

- Discussing how alignment preferences are communicated between two peers without using CCP

- Describing the treatment of RDMA_MSGP type messages that convey Read or Write chunks

The RDMA_MSGP message type is beneficial only when the padded data payload is at the end of an RPC message's argument or result list. This is not typical for NFSv4 COMPOUND RPCs, which often include a GETATTR operation as the final element of the compound operation array.

Without a full specification of RDMA_MSGP, there has been no fully implemented prototype of it. Without a complete prototype of RDMA_MSGP support, it is difficult to assess whether this protocol element has benefit, or can even be made to work interoperably.

Therefore, senders MUST NOT send RDMA_MSGP type messages. When receiving an RDMA_MSGP type message, receivers SHOULD reply with an RDMA_ERROR type message, setting the rdma_err field to ERR_CHUNK.

5.6.2. RDMA_DONE

Because no implementation of RPC-over-RDMA Version One uses the Read-Read transfer model, there is never a need to send an RDMA_DONE type message.

Therefore, senders MUST NOT send RDMA_DONE messages. When receiving an RDMA_DONE type message, receivers SHOULD reply with an RDMA_ERROR type message, setting the rdma_err field to ERR_CHUNK.

5.7. 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 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 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.

6. 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:

o 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.

o 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.

o 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 11, 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.

7. Bi-Directional RPC-Over-RDMA

7.1. RPC Direction

7.1.1. 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.

7.1.2. 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.

7.1.3. 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.

7.1.4. 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 represent distinct ONC RPC transactions.

7.2. Backward Direction Flow Control

7.2.1. 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.

7.2.2. 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 MAY drop the RDMA connection.

7.2.2.1. 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.

7.2.2.2. 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.

7.3. Conventions For Backward Operation

7.3.1. 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.

7.3.2. 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.

7.3.3. 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.

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.

7.3.4. 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.

7.3.5. 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.

7.4. 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 reduction, 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.

8. Upper Layer Binding Specifications

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

An Upper Layer Protocol is typically defined independently of a particular RPC transport. An Upper Layer Binding specification provides guidance that helps the Upper Layer Protocol interoperate correctly and efficiently over a particular transport, such as RPC-over-RDMA Version One. In particular, it provides:

- o A taxonomy of XDR data items that are eligible for Direct Data Placement

- o Clarifications on how to compute the maximum reply size for operations in the Upper Layer Protocol

- o An rpcbind port assignment for operation of the RPC Program and Version on an RPC-over-RDMA transport

8.1. DDP-Eligibility

To optimize the use of an RDMA transport, an Upper Layer Binding designates some XDR data items as eligible for Direct Data Placement. A data item is a candidate for eligibility if there is a clear benefit for moving the contents of the item directly from the sender's memory into the receiver's memory. Criteria for DDP-eligibility include:

1. The size of the XDR data item is frequently much larger than the inline threshold.

2. Transport-level processing of the XDR data item is not needed. For example, the data item is an opaque byte array, which requires no XDR encoding and decoding of its content.

3. The content of the XDR data item is sensitive to address alignment. For example, pullup would be required on the receiver before the content of the item can be used.

As RPC-over-RDMA messages are formed, DDP-eligible data items are treated specially. A DDP-eligible XDR data item is one that MAY be conveyed by itself in a separate chunk. The Upper Layer Protocol implementation or the RDMA transport implementation decides when to move a DDP-eligible data item into a chunk instead of leaving the item in the RPC message's XDR stream.

All other XDR data items are considered non-DDP-eligible, and MUST NOT be moved in a separate chunk. They MAY, however, be moved as part of a Position Zero Read Chunk or a Reply chunk.

The interface by which an Upper Layer implementation indicates the DDP-eligibility of a data item to the RPC transport is not described by this specification. The only requirements are that the receiver can re-assemble the transmitted RPC-over-RDMA message into a valid XDR stream, and that DDP-eligibility rules specified by the Upper Layer Binding are respected.

There is no provision to express DDP-eligibility within the XDR language. The only definitive specification of DDP-eligibility is the Upper Layer Binding itself.

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. Note however that such complex data types are unlikely to be good candidates for Direct Data Placement.

8.1.1. Write List Ordering Ambiguity

A requester constructs the Write list for an RPC transaction before the responder has formulated its reply. When there is only one result data item that is DDP-eligible, the requester appends only a single Write chunk to that Write list. If the responder populates that chunk with data, the requester knows with certainty which result is contained in it.

However, Upper Layer Protocol procedures may allow replies where more than one result data item is DDP-eligible. For example, an NFSv4 COMPOUND is composed of individual NFSv4 operations, more than one of which may have a reply containing a DDP-eligible result. As stated in Section 5.3.2, 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.

Ambiguities can arise when replies contain XDR unions or arrays of complex data types which allow a responder options about whether a DDP-eligible data item is included or not. It is the responsibility of the Upper Layer Binding to avoid situations where there is ambiguity about which result is in which chunk in the Write list. If an ambiguity is unavoidable, the Upper Layer Binding MUST specify how Write list entries are mapped to DDP-eligible results.

8.1.2. DDP-Eligibility Violation

A DDP-eligibility violation occurs when a requester forms a Call message with a non-DDP-eligible data item in a Read chunk, or provides a Write list when there are no DDP-eligible items allowed in the operation’s reply. A violation occurs when a responder forms a Reply message without reducing a DDP-eligible data item when there is a Write list provided by the requester.

In the first case, a responder might attempt to parse and process the Call message anyway. If the responder cannot process the Call, it MUST report this either via an RDMA_ERROR type message with the rdma_err field set to ERR_CHUNK, or via an RPC-level RPC_GARBAGEARGS message.

In the second case, the responder is in a bind: when a Write chunk is provided, it MUST use it, but the ULB specification does not say what result is expected in that chunk. This is considered a transport-level error, and MUST be reported to the requester via an RDMA_ERROR type message with the rdma_err field set to ERR_CHUNK.

In the third case, a requester might attempt to parse and process the Reply message anyway. If the requester cannot process the Reply, it MUST report this via an RDMA_ERROR type message with the rdma_err field set to ERR_CHUNK.

8.2. Maximum Reply Size

A requester provides resources for both a Call message and its matching Reply message. A requester forms the Call message itself, thus can compute the exact resources needed for it.

A requester must allocate resources for the Reply message (an RPC-over-RDMA credit, a Receive buffer, and possibly a Write list and Reply chunk) before the responder has formed the actual reply. To accommodate all possible replies for the operation in the Call message, a requester must allocate reply resources based on the maximum possible size of the expected reply.

If there are operations in the Upper Layer Protocol for which there is no clear payload maximum, an Upper Layer Binding MUST provide a mechanism a requester implementation can use to determine the resources needed for these operations.

8.3. Additional Considerations

There may be other details provided in an Upper Layer Binding.

- An Upper Layer Binding may recommend an inline threshold value or other transport-related parameters for RPC-over-RDMA Version One connections bearing that Upper Layer Protocol.

- An Upper Layer Protocol may provide a means to communicate these transport-related parameters between peers. Note that RPC-over-RDMA Version One does not specify any mechanism for changing any transport-related parameter after a connection has been established.

- Multiple Upper Layer Protocols may share a single RPC-over-RDMA Version One connection when their Upper Layer Bindings allow the use of RPC-over-RDMA Version One and the rpcbind port assignments for the Protocols allow connection sharing. In this case, the same transport parameters (such as inline threshold) apply to all Protocols using that connection.

Given the above, Upper Layer Bindings and Upper Layer Protocols must be designed to interoperate correctly no matter what connection parameters are in effect on a connection.

8.4. Upper Layer Protocol Extensions

An RPC Program and Version tuple may be extensible. For instance, there may be a minor versioning scheme that is not reflected in the RPC version number. Or, the Upper Layer Protocol may allow additional features to be specified after the original RPC program specification was ratified. Upper Layer Bindings are provided for interoperable programs and versions by extending existing Upper Layer Bindings to reflect the changes made necessary by each addition to the existing XDR.

9. Transport Protocol Extensibility

Upper Layer RPC Protocols 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.

- o 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.

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

- o 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.

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

9.1. RPC-over-RDMA Version Numbering

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

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

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

- o Whenever the set of allowable transfer models is altered

10. Security Considerations

10.1. Memory Protection

A primary consideration is the protection of the integrity and privacy of local memory by an RPC-over-RDMA transport. The use of RPC-over-RDMA MUST NOT introduce any vulnerabilities to system memory contents, nor to memory owned by user processes.

It is REQUIRED that any RDMA provider used for RPC transport be conformant to the requirements of [RFC5042] in order to satisfy these protections. These protections are provided by the RDMA layer specifications, and specifically their security models.

10.1.1. Protection Domains

The use of Protection Domains to limit the exposure of memory segments 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.

10.1.2. Handle Predictability

Unpredictable memory handles should be used for any operation requiring advertised memory segments. 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 implementations should avoid advertising persistently registered memory.

10.1.3. Memory Fencing

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

10.2. Using GSS With RPC-Over-RDMA

ONC 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 host data movement associated with the computation and verification of integrity and with encryption/decryption, so performance advantages can 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.

Once delivered securely by the RDMA provider, any RDMA-exposed memory 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.

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 6. 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].

Dave Noveck provided excellent review, constructive suggestions, and consistent navigational guidance throughout the process of drafting this document.

The comments and contributions of Karen Deitke, Dai Ngo, Chunli Zhang, Dominique Martinet, and Mahesh Siddheshwar are accepted with many and great thanks. The editor also wishes to thank Bill Baker for his 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. References

13.1. 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>.

13.2. 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>.

[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>.

[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

William Allen Simpson
DayDreamer
1384 Fontaine
Madison Heights, MI 48071
USA

Email: william.allen.simpson@gmail.com

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

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