Remote Direct Memory Access Transport for Remote Procedure Call, Version
                                  One
                    draft-ietf-nfsv4-rfc5666bis-05

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

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

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

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

   This Internet-Draft will expire on October 10, 2016.

Copyright Notice

to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. 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. It makes the role of Upper Layer
   Bindings an explicit part of the protocol specification.

   In addition, this document describes current practice using
   RPCSEC_GSS [I-D.ietf-nfsv4-rpcsec-gssv3] 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. Remote Procedure Calls 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
   [RFC0768] or TCP [RFC0793]. 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].

   o Section 2 has been expanded to introduce and explain key RPC, XDR,
      and RDMA terminology. These terms are now used consistently
      throughout the specification.

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

   o Sections 4 and 5 have been combined to improve the organization of
      this information.

   o The specification of the optional Connection Configuration
      Protocol has been removed from the specification.

   o A section consolidating requirements for Upper Layer Bindings has
      been added.

   o An XDR extraction mechanism is provided, along with full
      copyright, matching the approach used in [RFC5662].

   o The "Security Considerations" section has been expanded to include
      a discussion of how RPC-over-RDMA security depends on features of
      the underlying RDMA transport.

   o A subsection describing the use of RPCSEC_GSS with RPC-over-RDMA
      Version One has been added.

2.2. Changes To The Protocol

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

o 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. Removal simplifies explanatory
   text, and support for the RDMA_DONE procedure is no longer
   necessary.

o The specification of RDMA_MSGP in [RFC5666] and current
   implementations of it are incomplete. Even if completed, benefit
   for protocols such as NFSv4.0 [RFC7530] is doubtful. Therefore
   the RDMA_MSGP message type is no longer supported.

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

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

o Explicit guidance is provided for sizing Write chunks, managing
   multiple chunks in the Write list, and handling unused Write
   chunks.

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

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 highlights key elements of the Remote Procedure Call
   [RFC5531] and External Data Representation [RFC4506] protocols, upon
   which RPC-over-RDMA Version One is constructed. Strong grounding
   with these protocols is recommended before reading this document.

3.1.1. Upper Layer Protocols

   Remote Procedure Calls are an abstraction used to implement the
   operations of an "Upper Layer Protocol," or 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 (RPC) 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 zero (0) in the message's msg_type field.
      An arbitrary unique value is placed in the message's xid field in
      order to match this CALL message to a corresponding REPLY message.

   REPLY Message
      A REPLY message, or "Reply", reports the results of work requested
      by a Call. A Reply is designated by the value one (1) 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.

   The 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 RPC 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 complete, 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, requesters initiate transport
connections, while responders wait passively for incoming connection
requests.

3.1.4. External Data Representation

One cannot assume that all requesters and responders internally
represent data objects 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. The contents of such a data item are referred to as
"opaque data." XDR encoding places the content of opaque data items
directly into an XDR stream without altering it in any way. Upper
Layer Protocols or applications perform any needed data translation
in this case. Examples of opaque data items include the content of
files, or generic byte strings.

3.1.4.2. XDR Round-up

The number of octets in a variable-size opaque data item precedes
that item in an XDR 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
transports enable 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 a subsequent 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 only the Read-Write Transfer Model. Therefore

the use of the Read-Read Transfer Model within RPC-over-RDMA Version
One implementations is no longer supported. Other Transfer Models
may be used in future versions of RPC-over-RDMA.

4.2. 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.

RPC Payload Stream
   The "Payload stream" contains the encapsulated RPC message being
   transferred by this RPC-over-RDMA message. This stream always
   begins with the XID field of the encapsulated RPC message.

Transport Stream
   The "Transport stream" contains a 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.

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. Managing Receiver Resources

It is critical to provide RDMA Send flow control for an RDMA
connection. If no pre-posted receive buffer is large enough to
accept an incoming RDMA Send, the RDMA Send operation fails. If a
pre-posted receive buffer is not available to accept an incoming RDMA
Send, the RDMA Send operation can fail. 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.

The longevity of an RDMA connection requires that sending endpoints
respect the resource limits of peer receivers. To ensure messages
can be sent and received reliably, there are two operational
parameters for each connection.

4.3.1. Credit Limit

The number of pre-posted RDMA Receive operations is sometimes
referred to as a peer's "credit limit." 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 further detail.

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. 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.
RDMA 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.

4.3.2. Inline Threshold

A receiver's "inline threshold" value is the largest message size (in
octets) that the receiver can accept via an RDMA Receive operation.
Each connection has two inline threshold values, one for each peer
receiver.

Unlike credit limits, 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.

4.3.3. Initial Connection State

When a connection is first established, peers might not know how many
receive buffers the other has, nor how large these 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 the use of RDMA 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

When a direct data placement capability is available, during XDR
encoding it can be determined that the transport can efficiently
place the content of one or more data items directly in the
receiver's memory, separately from the transfer of other parts of the
containing XDR stream.

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 can be split out and moved via
   separate RDMA operations. The piece of memory containing that
   portion of the data stream and metadata in an RPC-over-RDMA header
   together comprise what is referred to as a "chunk." A Payload stream
   after chunks have been removed is referred to as a "reduced" Payload
   stream. Likewise, a data item that has been removed from a Payload
   stream to be transferred separately is referred to as a "reduced"
   data item.

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 independent of RPC-over-RDMA, and thus is not specified by
   this document.

   To maintain interoperability on an RPC-over-RDMA transport, a
   determination must be made 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 7.

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 specified region of the requester's memory.

   An "RDMA segment", or a "plain 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 |
      +-+
      | Length |
      +-+
      | |
      +   Offset   +
      | |
      +-+
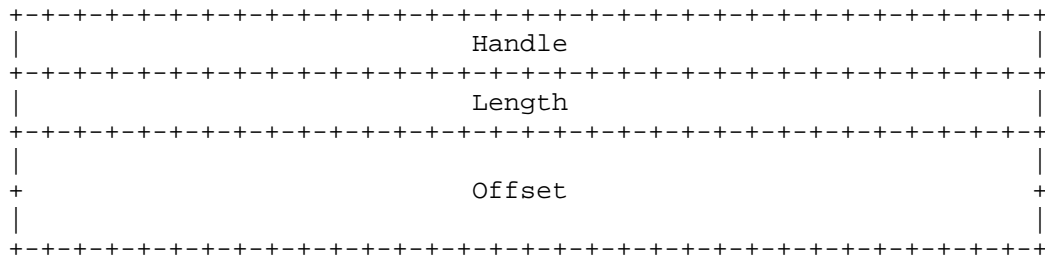
   Handle
      Steering tag (STag) or handle obtained when the segment's memory
      is registered for RDMA. Also known as an R_key, this value is
      generated by registering this memory with the RDMA provider.

   Length
      The length of the memory segment, in octets.

   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. Segments MAY
   divide a chunk on any boundary that is convenient to the requester.

   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 RPC-over-RDMA 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,
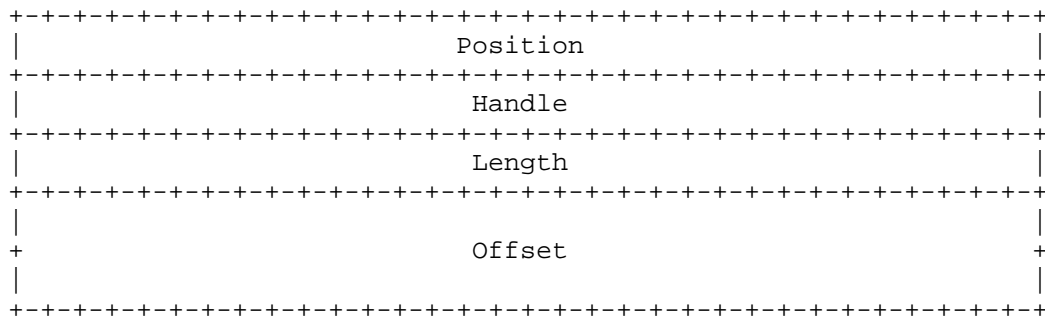   MUST be moved to the chunk.

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 is a plain segment which has an additional
   Position field.

```
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                            Position                           |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                             Handle                            |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                             Length                            |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 |                                                               |
 +                             Offset                            +
 |                                                               |
 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   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 RDMA 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, the responder 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 encoded, 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 read
   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. If the requester provides no actual round-up in a Read
   chunk, the responder MUST be prepared to provide appropriate round-up
   in the reconstructed call XDR stream

   The Position field in a read segment indicates where the containing
   Read chunk starts in the Payload 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

   While decoding a received Payload stream, whenever the XDR offset in
   the Payload stream matches that of a Read chunk, the transport
   initiates an RDMA Read to pull the chunk's data content into
   registered memory on the responder.

   The responder acknowledges its completion of use of Read chunk source
   buffers when it sends an RPC Reply 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 plain RDMA segments. Write
   chunks are provided by a requester long before the responder has
   prepared the reply Payload stream. Therefore RDMA segments in a
   Write chunk do not have a Position field.

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 MUST NOT change the number of segments in the Write chunk.

The responder fills the segments in array order until the data item
has been completely written. The responder updates the segment
length fields to reflect the actual amount of data that is being
returned in each segment. If a Write chunk segment is not filled by
the responder, the updated length of the segment SHOULD be zero.

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. Write Chunk Round-up

XDR requires each encoded data item to start on four-byte alignment.
When an odd-length data item is encoded, 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.

4.4.6.2. 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 while other
arms do not. The requester is required to provide a Write chunk in
this case, but if the responder returns a result that uses an arm of

the union that has no DDP-eligible data item, the Write chunk remains
unused.

When forming an RPC-over-RDMA Reply message with 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. 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.

4.5. 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. However, not all RPC messages are 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.

An RPC-over-RDMA transaction using Short Messages:

```
        Requester                                Responder
            |          RDMA Send (RDMA_MSG)         |
      Call  |     ------------------------------>   |
            |                                       | Processing
            |                                       |
            |                                       |
            |          RDMA Send (RDMA_MSG)         |
            |     <------------------------------   | Reply
```
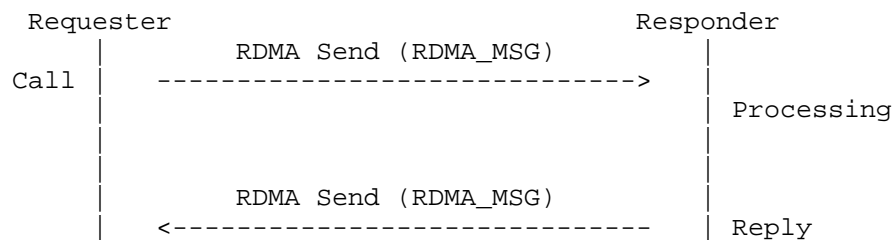
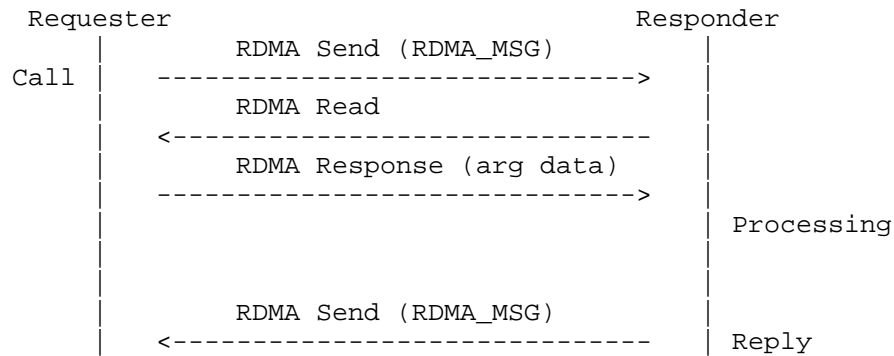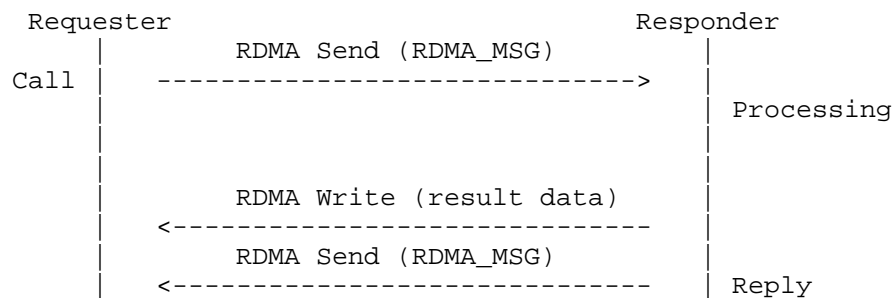4.5.2. Chunked Messages

   If DDP-eligible data items are present in a Payload stream, a sender
   MAY reduce some or all of these items by removing them from the
   Payload stream. The sender uses RDMA Read or Write operations to
   transfer the reduced data items. The Transport stream with the
   reduced Payload stream immediately following is then 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.

   An RPC-over-RDMA transaction with a Read chunk:

          Requester   Responder
              | RDMA Send (RDMA_MSG) |
         Call | -----------------------------> |
              | RDMA Read |
              | <----------------------------- |
              | RDMA Response (arg data) |
              | -----------------------------> |
              | | Processing
              | |
              | |
              | RDMA Send (RDMA_MSG) |
              | <----------------------------- | Reply

   An RPC-over-RDMA transaction with a Write chunk:

          Requester   Responder
              | RDMA Send (RDMA_MSG) |
         Call | -----------------------------> |
              | | Processing
              | |
              | |
              | RDMA Write (result data) |
              | <----------------------------- |
              | RDMA Send (RDMA_MSG) |
              | <----------------------------- | Reply

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."

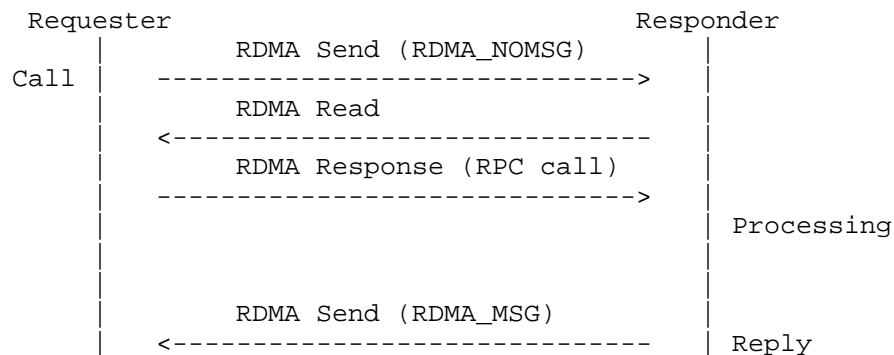   Long RPC Reply
      To send a Long RPC-over-RDMA Reply message, the requester provides
      a single special Write chunk in advance, known as the "Reply
      chunk", that will contain the RPC Reply's Payload stream. The
      requester sizes the Reply chunk to accommodate the maximum
      expected reply size for that Upper Layer operation.

   Though the purpose of a Long Message is to handle large RPC messages,
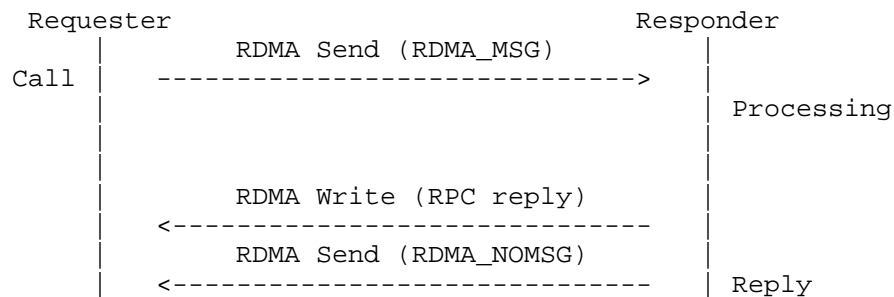   requesters MAY use a Long Message at any time to convey an RPC Call.

   A responder chooses which form of reply to use based on the chunks
   provided by the requester. If Write chunks were provided and the
   responder has a DDP-eligible result, it first reduces the reply
   Payload stream. If a Reply chunk was provided and the reduced
   Payload stream is larger than the requester's inline threshold, the
   responder MUST use the provided Reply chunk for the reply.

   Because these special chunks contain a whole RPC message, XDR data
   items appear in these special chunks without regard to their DDP-
   eligibility.

   An RPC-over-RDMA transaction using a Long Call:

```
         Requester                             Responder
             |          RDMA Send (RDMA_NOMSG)     |
        Call |    ------------------------------>  |
             |          RDMA Read                  |
             |    <------------------------------  |
             |          RDMA Response (RPC call)   |
             |    ------------------------------>  |
             |                                     | Processing
             |                                     |
             |                                     |
             |          RDMA Send (RDMA_MSG)       |
             |    <------------------------------  | Reply
```

   An RPC-over-RDMA transaction using a Long Reply:

```
         Requester                             Responder
             |          RDMA Send (RDMA_MSG)        |
        Call |    ------------------------------>   |
             |                                      | Processing
             |                                      |
             |                                      |
             |          RDMA Write (RPC reply)      |
             |    <------------------------------   |
             |          RDMA Send (RDMA_NOMSG)      |
             |    <------------------------------   | Reply
```

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.

   The RPC-over-RDMA protocol passes RPC messages without regard to
   their type (CALL or REPLY) or direction (forwards or backwards).
   Each endpoint of a connection MAY send any RPC-over-RDMA message
   header type at any time (subject to credit limits).

5.1. XDR Protocol Definition

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

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

   <CODE BEGINS>

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

   <CODE ENDS>

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

   <CODE BEGINS>

```
sh extract.sh < spec.txt > rpcrdma_corev1.x
```

   <CODE ENDS>

5.1.1. Code Component License

   Code components extracted from this document must include the
   following license text. When the extracted XDR code is combined with
   other complementary XDR code which itself has an identical license,
   only a single copy of the license text need be preserved.

```
    <CODE BEGINS>

    /// /*
    ///  * Copyright (c) 2010, 2016 IETF Trust and the persons
    ///  * identified as authors of the code.  All rights reserved.
    ///  *
    ///  * The authors of the code are:
    ///  * B. Callaghan, T. Talpey, 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.
    ///  */
    ///

    <CODE ENDS>
```

5.1.2. RPC-Over-RDMA Version One XDR

   XDR data items defined in this section encodes the Transport Header
   Stream in each RPC-over-RDMA Version One message. Comments identify
   items that cannot be changed in subsequent versions.

   <CODE BEGINS>

```
/// /*
///  * Plain RDMA segment (Section 4.4.3)
///  */
/// struct xdr_rdma_segment {
///    uint32 handle;          /* Registered memory handle */
///    uint32 length;          /* Length of the chunk in bytes */
///    uint64 offset;          /* Chunk virtual address or offset */
/// };
///
/// /*
///  * Read segment (Section 4.4.5)
///  */
/// struct xdr_read_chunk {
///    uint32 position;        /* Position in XDR stream */
///    struct xdr_rdma_segment target;
/// };
///
/// /*
///  * Read list (Section 5.3.1)
///  */
/// struct xdr_read_list {
///         struct xdr_read_chunk entry;
///         struct xdr_read_list  *next;
/// };
///
/// /*
///  * Write chunk (Section 4.4.6)
///  */
/// struct xdr_write_chunk {
///         struct xdr_rdma_segment target<>;
/// };
///
/// /*
///  * Write list (Section 5.3.2)
///  */
/// struct xdr_write_list {
///         struct xdr_write_chunk entry;
///         struct xdr_write_list  *next;
/// };
```

```
///
/// /*
///  * Chunk lists (Section 5.3)
///  */
/// struct rpc_rdma_header {
///    struct xdr_read_list   *rdma_reads;
///    struct xdr_write_list  *rdma_writes;
///    struct xdr_write_chunk *rdma_reply;
///    /* rpc body follows */
/// };
///
/// struct rpc_rdma_header_nomsg {
///    struct xdr_read_list   *rdma_reads;
///    struct xdr_write_list  *rdma_writes;
///    struct xdr_write_chunk *rdma_reply;
/// };
///
/// struct rpc_rdma_header_padded {
///    uint32                 rdma_align;    /* Padding alignment */
///    uint32                 rdma_thresh;   /* Padding threshold */
///    struct xdr_read_list   *rdma_reads;
///    struct xdr_write_list  *rdma_writes;
///    struct xdr_write_chunk *rdma_reply;
///    /* rpc body follows */
/// };
///
/// /*
///  * Error handling (Section 5.5)
///  */
/// enum rpc_rdma_errcode {
///    ERR_VERS = 1,       /* Fixed for all versions */
///    ERR_CHUNK = 2
/// };
///
/// struct rpc_rdma_errvers {
///    uint32 rdma_vers_low;
///    uint32 rdma_vers_high;
/// };
///
/// union rpc_rdma_error switch (rpc_rdma_errcode err) {
///    case ERR_VERS:
///      rpc_rdma_errvers range;
///    case ERR_CHUNK:
///      void;
/// };
///
/// /*
///  * Procedures (Section 5.2.4)
```

```
///   */
/// enum rdma_proc {
///     RDMA_MSG = 0,      /* Fixed for all versions */
///     RDMA_NOMSG = 1,    /* Fixed for all versions */
///     RDMA_MSGP = 2,     /* Reserved */
///     RDMA_DONE = 3,     /* Reserved */
///     RDMA_ERROR = 4     /* Fixed for all versions */
/// };
///
/// union rdma_body switch (rdma_proc proc) {
///     case RDMA_MSG:
///       rpc_rdma_header rdma_msg;
///     case RDMA_NOMSG:
///       rpc_rdma_header_nomsg rdma_nomsg;
///     case RDMA_MSGP:
///       rpc_rdma_header_padded rdma_msgp;
///     case RDMA_DONE:
///       void;
///     case RDMA_ERROR:
///       rpc_rdma_error rdma_error;
/// };
///
/// /*
///  * Fixed header fields (Section 5.2)
///  */
/// struct rdma_msg {
///     uint32    rdma_xid;
///     uint32    rdma_vers;
///     uint32    rdma_credit;
///     rdma_body rdma_body;
/// };

<CODE ENDS>
```

5.2. Fixed Header Fields

   The RPC-over-RDMA header begins with four fixed 32-bit fields that
   control the RDMA interaction. These four fields, which must remain
   with the same meanings and in the same positions in all subsequent
   versions of the RPC-over-RDMA protocol, are described below.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             XID                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Version Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Credit Value                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Procedure Number                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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 each RPC-over-RDMA 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 one
(1). Rules regarding changes to this transport protocol version
number can be found in Section 8.

5.2.3. 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 how the credit value
is determined can be found in Section 4.3.

5.2.4. Procedure number

o RDMA_MSG = 0 indicates that chunk lists and a Payload stream
   follow. The format of the chunk lists is discussed below.

o RDMA_NOMSG = 1 indicates that after the chunk lists there is no
   Payload stream. In this case, the chunk lists provide information
   to allow the responder to transfer the Payload stream 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 encoding error in the RPC-
   over-RDMA header.

An RDMA_MSG procedure 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 procedure 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 of these MAY be marked as not present, one MUST be present
and MUST hold the Payload stream for this RPC-over-RDMA message. 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 procedure 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 combine
the Transport and Payload streams, which might have been constructed
in separate buffers. 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 follow the fixed header fields in RDMA_MSG
   and RDMA_NOMSG procedures. 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 procedure 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 argument data the
   responder is to retrieve via RDMA Read operations. The requester has
   removed the data in these chunks from the call's Payload stream.

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

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

Responders MUST leave the Read list empty in all replies.

5.3.2. Write List

Each RDMA_MSG or RDMA_NOMSG procedure 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. If the RPC Reply has no possible DDP-eligible result data items, the requester leaves the Write list empty.

When a Write list is provided for the results of an RPC Call, the responder MUST provide data corresponding to DDP-eligible XDR data items via RDMA Write operations to the memory referenced in the Write list. The responder removes the data in these chunks from the reply's Payload stream.

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 requester may not be able to predict which DDP-eligible data item goes in which chunk. Thus the requester is responsible for allocating and registering Write chunks large enough to accommodate the largest XDR data item that might be associated with each chunk in the list.

The RPC Reply conveys the size of result data items by returning each Write chunk to the requester with the segment lengths rewritten to match the actual data transferred. Decoding the reply therefore performs no local data copying 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 Write list entry is consumed.

A requester constructs the Write list for an RPC transaction before the responder has formulated its reply. When there is only one DDP-eligible result data item, the requester inserts only a single Write

chunk in the Write list. If the responder populates that chunk with
data, the requester knows with certainty which result data item 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 procedure is composed of individual NFSv4 operations, more
than one of which may have a reply containing a DDP-eligible result.

As stated above, when multiple Write chunks are present, the
responder reduces DDP-eligible results until either there are no more
results or no more Write chunks. Then, as the requester decodes the
reply Payload stream, it is clear from the contents of the reply
which Write chunk contains which data item.

When a requester has provided a Write list in a Call message, the
responder MUST copy that list into the associated Reply. The copied
Write list in the Reply is modified as above to reflect the actual
amount of data that is being returned in the Write list.

5.3.3. Reply Chunk

Each RDMA_MSG or RDMA_NOMSG procedure 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. If the actual reply Payload stream is smaller
than the requester's inline threshold, the responder MAY return it as
a Short message rather than using the Reply chunk.

When a requester has provided a Reply chunk in a Call message, the
responder MUST copy that chunk into the associated Reply. The copied
Reply chunk in the Reply is modified to reflect the actual amount of
data that is being returned in the Reply 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. For memory that is targeted by

RDMA Send and Receive operations, a local-only registration is
sufficient and can be left in place during the life of a connection
without any risk of data exposure.

5.4.1. Registration Longevity

Data transferred via RDMA Read and Write can reside in a memory
allocation not in the control of the RPC-over-RDMA transport. These
memory allocations can persist outside the bounds of an RPC
transaction. They are registered and invalidated as needed, as part
of each RPC transaction.

The requester endpoint must ensure that memory segments associated
with each RPC transaction are properly fenced from responders before
allowing Upper Layer access to the data contained in them. Moreover,
the requester must not access these memory segments while the
responder has access to them.

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 might have 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 procedure MUST be generated. Because the transport layer
may not be aware of the direction of a problematic RPC message, an
RDMA_ERROR procedure MAY be generated by either a requester or a
responder.

To form an RDMA_ERROR procedure: 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 procedure indicates a permanent error. Receipt of this
procedure completes the RPC transaction associated with XID in the
rdma_xid field. A receiver MUST silently discard an RDMA_ERROR
procedure that it cannot decode.

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_ERROR procedure and set the rdma_err value to
ERR_VERS, also 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. If the rdma_vers field contains a
recognized value, but an XDR parsing error occurs, the responder MUST
reply with an RDMA_ERROR procedure and set the rdma_err value to
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 procedure. 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 RDMA Operational Errors

   In RPC-over-RDMA Version One, it is the responder which drives RDMA
   Read and Write operations that target the requester's memory.
   Problems might arise as the responder attempts to use requester-
   provided resources for RDMA operations. For example:

   o 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.

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

   o 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.

   RDMA 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. Other Operational Errors

   While a requester is constructing a Call message, an unrecoverable
   problem might occur that prevents the requester from posting further
   RDMA Work Requests on behalf of that message. As with other
   transports, if a requester is unable to construct and transmit a Call
   message, the associated RPC transaction fails immediately.

After a requester has received a reply, if it is unable to invalidate
a memory region due to an unrecoverable problem, the requester MUST
close the connection to fence that memory from the responder before
the associated RPC transaction is complete.

While a responder is constructing a Reply message or error message,
an unrecoverable problem might occur that prevents the responder from
posting further RDMA Work Requests on behalf of that message. If a
responder is unable to construct and transmit a Reply or error
message, the responder MUST close the connection to signal to the
requester that a reply was lost.

5.5.5. 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
transport, and recover via retransmission. RPC recovery can handle
complete loss and re-establishment of a transport connection.

The details of reporting and recovery from RDMA link layer errors are
outside the scope of this protocol specification. See Section 9 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:

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

o Adding full operational descriptions of the alignment and
   threshold fields

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

o Describing the treatment of RDMA_MSGP procedures 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 procedures. When
receiving an RDMA_MSGP procedure, receivers SHOULD reply with an
RDMA_ERROR procedure, 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
procedure.

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

5.7. XDR Examples

RPC-over-RDMA chunk lists are complex data types. In this section,
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 10, 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. Upper Layer Binding Specifications

An Upper Layer Protocol is typically defined independently of any
particular RPC transport. An Upper Layer Binding specification (ULB)
provides guidance that helps the Upper Layer Protocol interoperate
correctly and efficiently over a particular transport. For RPC-over-
RDMA Version One, an Upper Layer Binding may provide:

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

o Constraints on which Upper Layer procedures may be reduced, and on
   how many chunks may appear in a single RPC request

o A method for determining the maximum size of the reply Payload
   stream for all procedures in the Upper Layer Protocol

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

Each RPC Program and Version tuple that utilizes RPC-over-RDMA
Version One needs to have an Upper Layer Binding specification.

7.1. DDP-Eligibility

   An Upper Layer Binding designates some XDR data items as eligible for
   Direct Data Placement. As an RPC-over-RDMA message is formed, DDP-
   eligible data items can be removed from the Payload stream and placed
   directly in the receiver's memory.

   An XDR data item should be considered for DDP-eligibility if there is
   a clear benefit to moving the contents of the item directly from the
   sender's memory to the receiver's memory. Criteria for DDP-
   eligibility include:

   o The XDR data item is frequently sent or received, and its size is
      often much larger than typical inline thresholds.

   o 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.

   o 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.

   o The XDR data item does not contain DDP-eligible data items.

   In addition to defining the set of data items that are DDP-eligible,
   an Upper Layer Binding may also limit the use of chunks to particular
   Upper Layer procedures. If more than one data item in a procedure is
   DDP-eligible, the Upper Layer Binding may also limit the number of
   chunks that a requester can provide for a particular Upper Layer
   procedure.

   Senders MUST NOT reduce data items that are not DDP-eligible. Such
   data items MAY, however, be moved as part of a Position Zero Read
   Chunk or a Reply chunk.

   The programming 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 an
   Upper Layer Binding.

7.1.1. 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. 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 MUST NOT process the Call message.

   In the second case, as a requester parses a Reply message, it must
   assume that the responder has correctly reduced a DDP-eligible result
   data item. If the responder has not done so, it is likely that the
   requester cannot finish parsing the Payload stream and that an XDR
   error would result.

   Both types of violations MUST be reported as described in
   Section 5.5.2.

7.2. 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 procedure in the Call
   message, a requester must allocate reply resources based on the
   maximum possible size of the expected Reply message.

   If there are procedures in the Upper Layer Protocol for which there
   is no clear reply size maximum, the Upper Layer Binding needs to
   specify a dependable means for determining the maximum.

7.3. Additional Considerations

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

   o 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.

   o 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.

o 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.

Each Upper Layer Binding needs to be designed to allow correct
interoperation without regard to the transport parameters actually in
use. Furthermore, implementations of Upper Layer Protocols must be
designed to interoperate correctly regardless of the connection
parameters in effect on a connection.

7.4. 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 RPC Programs and
Versions by extending existing Upper Layer Bindings to reflect the
changes made necessary by each addition to the existing XDR.

8. Protocol Extensibility

The RPC-over-RDMA header format is specified using XDR, unlike the
message header used with RPC over TCP. To maintain a high degree of
interoperability among implementations of RPC-over-RDMA, any change
to this XDR requires a protocol version number change. New versions
of RPC-over-RDMA may be published as separate protocol specifications
without updating this document.

The first four fields in every RPC-over-RDMA header must remain
aligned at the same fixed offsets for all versions of the RPC-over-
RDMA protocol. The version number must be in a fixed place to enable
implementations to detect protocol version mismatches.

For version mismatches to be reported in a fashion that all future
version implementations can reliably decode, the rdma_proc field must
remain in a fixed place, the value of ERR_VERS must always remain the
same, and the field placement in struct rpc_rdma_errvers must always
remain the same.

8.1. Conventional Extensions

   Introducing new capabilities to RPC-over-RDMA Version One is limited
   to the adoption of conventions that make use of existing XDR (defined
   in this document) and allowed abstract RDMA operations. Because no
   mechanism for detecting optional features exists in RPC-over-RDMA
   Version One, implementations must rely on Upper Layer Protocols to
   communicate the existence of such extensions.

   Such extensions must be specified in a Standards Track document with
   appropriate review by the nfsv4 Working Group and the IESG. An
   example of a conventional extension to RPC-over-RDMA Version One can
   be found in [I-D.ietf-nfsv4-rpcrdma-bidirection].

9. Security Considerations

9.1. 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 in particular, their security models.

9.1.1. Protection Domains

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

9.1.2. 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.

9.1.3. Memory Fencing

   Requesters should register memory segments for remote access only
   when they are about to be the target of an RPC operation that
   involves an RDMA Read or Write.

   Registered memory segments should be invalidated as soon as related
   RPC operations are complete. Invalidation and DMA unmapping of RDMA
   segments should be complete before message integrity checking is
   done, and before the RPC consumer is allowed to continue execution
   and use or alter the contents of a memory region.

   An RPC transaction on a requester might be terminated before a reply
   arrives if the RPC consumer exits unexpectedly (for example it is
   signaled or a segmentation fault occurs). When an RPC terminates
   abnormally, memory segments associated with that RPC should be
   invalidated appropriately before the segments are released to be
   reused for other purposes on the requester.

9.2. RPC Message Security

   ONC RPC provides cryptographic security via the RPCSEC_GSS framework
   [I-D.ietf-nfsv4-rpcsec-gssv3]. RPCSEC_GSS implements message
   authentication, per-message integrity checking, and per-message
   confidentiality. However, integrity and privacy services require
   significant movement of data on each endpoint host. Some performance
   benefits enabled by RDMA transports can be lost.

9.2.1. RPC-Over-RDMA Protection At Lower Layers

   Note that performance loss is expected when RPCSEC_GSS integrity or
   privacy is in use on any RPC transport. Protection below the RDMA
   layer is a more appropriate security mechanism for RDMA transports in
   performance-sensitive deployments. Certain configurations of IPsec
   can be co-located in RDMA hardware, for example, without any change
   to RDMA consumers or loss of data movement efficiency.

   The use of protection in a lower layer MAY be negotiated through the
   use of an RPCSEC_GSS security flavor defined in
   [I-D.ietf-nfsv4-rpcsec-gssv3] 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.

9.2.2. RPCSEC_GSS On RPC-Over-RDMA Transports

   Not all RDMA devices and fabrics support the above protection
   mechanisms. Also, per-message authentication is still required on
   NFS clients where multiple users access NFS files. In these cases,
   RPCSEC_GSS can protect NFS traffic conveyed on RPC-over-RDMA
   connections.

   RPCSEC_GSS extends the ONC RPC protocol [RFC5531] without changing
   the format of RPC messages. By observing the conventions described
   in this section, an RPC-over-RDMA transport can convey RPCSEC_GSS-
   protected RPC messages interoperably.

   As part of the ONC RPC protocol, protocol elements of RPCSEC_GSS that
   appear in the Payload stream of an RPC-over-RDMA message (such as
   control messages exchanged as part of establishing or destroying a
   security context, or data items that are part of RPCSEC_GSS
   authentication material) MUST NOT be reduced.

9.2.2.1. RPCSEC_GSS Context Negotiation

   Some NFS client implementations use a separate connection to
   establish a GSS context for NFS operation. These clients use TCP and
   the standard NFS port (2049) for context establishment. However
   there is no guarantee that an NFS/RDMA server provides a TCP-based
   NFS server on port 2049.

9.2.2.2. RPC-Over-RDMA With RPCSEC_GSS Authentication

   The RPCSEC_GSS authentication service has no impact on the DDP-
   eligibility of data items in an Upper Layer Protocol.

   However, RPCSEC_GSS authentication material appearing in an RPC
   message header can be larger than, say, an AUTH_SYS authenticator.
   In particular, when an RPCSEC_GSS pseudoflavor is in use, a requester
   needs to accommodate a larger RPC credential when marshaling Call
   messages, and to provide for a maximum size RPCSEC_GSS verifier when
   allocating reply buffers and Reply chunks.

   RPC messages, and thus Payload streams, are made larger as a result.
   Upper Layer Protocol operations that fit in a Short Message when a
   simpler form of authentication is in use might need to be reduced, or
   conveyed via a Long Message, when RPCSEC_GSS authentication is in
   use. It is more likely that a requester provides both a Read list
   and a Reply chunk in the same RPC-over-RDMA header to convey a Long
   call and provision a receptacle for a Long reply. More frequent use
   of Long messages can impact transport efficiency.

9.2.2.3. RPC-Over-RDMA With RPCSEC_GSS Integrity Or Privacy

   The RPCSEC_GSS integrity service enables endpoints to detect
   modification of RPC messages in flight. The RPCSEC_GSS privacy
   service prevents all but the intended recipient from viewing the
   cleartext content of RPC arguments and results. RPCSEC_GSS integrity
   and privacy are end-to-end. They protect RPC arguments and results
   from application to server endpoint, and back.

   The RPCSEC_GSS integrity and encryption services operate on whole RPC
   messages after they have been XDR encoded for transmit, and before
   they have been XDR decoded after receipt. Both sender and receiver
   endpoints use intermediate buffers to prevent exposure of encrypted
   data or unverified cleartext data to RPC consumers. After
   verification, encryption, and message wrapping has been performed,
   the transport layer MAY use RDMA data transfer between these
   intermediate buffers.

   The process of reducing a DDP-eligible data item removes the data
   item and its XDR padding from the encoded XDR stream. XDR padding of
   a reduced data item is not transferred in an RPC-over-RDMA message.
   After reduction, the Payload stream contains fewer octets then the
   whole XDR stream did beforehand. XDR padding octets are often zero
   bytes, but they don't have to be. Thus reducing DDP-eligible items
   affects the result of message integrity verification or encryption.

   Therefore a sender MUST NOT reduce a Payload stream when RPCSEC_GSS
   integrity or encryption services are in use. Effectively, no data
   item is DDP-eligible in this situation, and Chunked Messages cannot
   be used. In this mode, an RPC-over-RDMA transport operates in the
   same manner as a transport that does not support direct data
   placement.

   When RPCSEC_GSS integrity or privacy is in use, a requester provides
   both a Read list and a Reply chunk in the same RPC-over-RDMA header
   to convey a Long call and provision a receptacle for a Long reply.

9.2.2.4. Protecting RPC-Over-RDMA Transport Headers

   Like the base fields in an ONC RPC message (XID, call direction, and
   so on), the contents of an RPC-over-RDMA message's Transport stream
   are not protected by RPCSEC_GSS. This exposes XIDs, connection
   credit limits, and chunk lists (but not the content of the data items
   they refer to) to malicious behavior, which could redirect data that
   is transferred by the RPC-over-RDMA message, result in spurious
   retransmits, or trigger connection loss.

In particular, if an attacker alters the information contained in the
chunk lists of an RPC-over-RDMA header, data contained in those
chunks can be redirected to other registered memory segments on
requesters. An attacker might alter the arguments of RDMA Read and
RDMA Write operations on the wire to similar effect. The use of
RPCSEC_GSS integrity or privacy services enable the requester to
detect if such tampering has been done and reject the RPC message.

Encryption at lower layers, as described in Section 9.2.1, protects
the content of the Transport stream. To address attacks on RDMA
protocols themselves, RDMA transport implementations should conform
to [RFC5042].

10. IANA Considerations

Three assignments are specified by this document. These are
unchanged from [RFC5666]:

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

o Optional service port assignments for Upper Layer Bindings

o 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 3.2.2, 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].

11. 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. Dave also contributed much of the organization and
content of Section 8 and helped the authors understand the
complexities of XDR extensibility.

The comments and contributions of Karen Deitke, Dai Ngo, Chunli
Zhang, Dominique Martinet, and Mahesh Siddheshwar are accepted with
great thanks. The editor also wishes to thank Bill Baker, Greg
Marsden, and Matt Benjamin for their support of this work.

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

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

12. References

12.1. Normative References

   [I-D.ietf-nfsv4-rpcrdma-bidirection]
              Lever, C., "Size-Limited Bi-directional Remote Procedure
              Call On Remote Direct Memory Access Transports", draft-
              ietf-nfsv4-rpcrdma-bidirection-01 (work in progress),
              September 2015.

   [I-D.ietf-nfsv4-rpcsec-gssv3]
              Adamson, A. and N. Williams, "Remote Procedure Call (RPC)
              Security Version 3", draft-ietf-nfsv4-rpcsec-gssv3-17
              (work in progress), January 2016.

   [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>.

   [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>.

   [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>.

12.2. Informative References

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

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

   [RFC0768]   Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI
               10.17487/RFC0768, August 1980,
               <http://www.rfc-editor.org/info/rfc768>.

   [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>.

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

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

   [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