Internet-Draft
Internet-Draft
Updates: 5040, 7306 (if approved)
Intended status: Standards Track
Expires: August 22, 2016

T. Talpey
J. Pinkerton
Microsoft

February 19, 2016

RDMA Durable Write Commit
draft-talpey-rdma-commit-00

Abstract

This document specifies extensions to RDMA protocols to provide capabilities in support of enhanced remotely-directed data consistency. The extensions include a new operation supporting remote commitment to durability of remotely-managed buffers, which can provide enhanced guarantees and improve performance for low-latency storage applications. In addition to, and in support of these, extensions to local behaviors are described, which may be used to guide implementation, and to ease adoption. This document would extend the IETF Remote Direct Memory Access Protocol (RDMAP), RFC5040, and RDMA Protocol Extensions, RFC7306.

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 RFC 2119 [RFC2119].

Status of This Memo

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

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

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

This Internet-Draft will expire on August 22, 2016.

Table of Contents

1. Introduction

The RDMA Protocol (RDMAP) [RFC5040] and RDMA Protocol Extensions (RDMAPEXT) [RFC7306] provide capabilities for secure, zero copy data communications that preserve memory protection semantics, enabling more efficient network protocol implementations. The RDMA Protocol is part of the iWARP family of specifications which also include the Direct Data Placement Protocol (DDP) [RFC5041], and others as described in the relevant documents. For additional background on RDMA Protocol applicability, see "Applicability of Remote Direct Memory Access Protocol (RDMA) and Direct Data Placement Protocol (DDP)" RFC5045 [RFC5045].

RDMA protocols are enjoying good success in improving the performance of remote storage access, and have been well-suited to semantics and latencies of existing storage solutions. However, new storage solutions are emerging with much lower latencies, driving new workloads and new performance requirements. Also, storage programming paradigms SNIANVM [SNIANVM] are driving new requirements of the remote storage layers, in addition to driving down latency tolerances. Overcoming these latencies, and providing the means to achieve durability without invoking upper layers and remote CPUs for each such request, are the motivators for the extensions proposed by this document.

This document specifies the following extensions to the RDMA Protocol (RDMAP) and its local memory ecosystem:

- RDMA Commit - support for RDMA requests and responses with enhanced placement semantics.

- Enhanced memory registration semantics in support of durability.

The extensions defined in this document do not require the RDMAP version to change.

1.1. Glossary

This document is an extension of RFC 5040 and RFC 7306, and key words are additionally defined in the glossaries of the referenced documents.

The following additional terms are defined in this document.

Commit: The placement of data into storage referenced by a target Tagged Buffer in a durable fashion.

Durability: The property that data is present and remains stable after recovery from a power failure or other fatal error in an upper layer or hardware. <https://en.wikipedia.org/wiki/Durability_(database_systems)>, <https://en.wikipedia.org/wiki/Disk_buffer#Cache_control_from_the_host>[SCSI],

2. Problem Statement

RDMA is widely deployed in support of storage and shared memory over increasingly low-latency and high-bandwidth networks. The state of the art today yields end-to-end network latencies on the order of one to two microseconds for message transfer, and bandwidths exceeding 40 gigabit/s. These bandwidths are expected to increase over time, with latencies decreasing as a direct result.

In storage, another trend is emerging - greatly reduced latency of persistently storing data blocks. While best-of-class Hard Disk Drives (HDDs) have delivered latencies of several milliseconds for many years, Solid State Disks (SSDs) have improved this by one to two orders of magnitude. Technologies such as NVM Express NVMe [1] yield even higher-performing results by eliminating the traditional storage interconnect. The latest technologies providing memory-based persistence, such as Nonvolatile Memory DIMM NVDIMM [2], places storage-like semantics directly on the memory bus, reducing latency to less than a microsecond and bandwidth to potentially many tens of gigabyte/s. [supporting data to be added]

RDMA protocols, in turn, are used for many storage protocols, including NFS/RDMA RFC5661 [RFC5661] RFC5666 [RFC5666] RFC5667

[RFC5667], SMB Direct MS-SMB2 [SMB3] MS-SMBD [SMBDirect] and iSER RFC7145 [RFC7145], to name just a few. These protocols allow storage and computing peers to take full advantage of these highly performant networks and storage technologies to achieve remarkable throughput, while minimizing the CPU overhead needed to drive their workloads. This leaves more computing resources available for the applications, which in turn can scale to even greater levels. Within the context of Cloud-based environments, and through scale-out approaches, this can directly reduce the number of servers that need to be deployed, making such attributes compelling.

However, limiting factors come into play when deploying ultra-low latency storage in such environments:

- The latency of the fabric, and of the necessary RDMA message exchanges to ensure reliable transfer is now higher than that of the storage itself.

- The requirement that storage be resilient to failure requires that multiple copies be committed in multiple locations across the fabric, adding extra hops which increase the latency and computing demand placed on implementing the resiliency.

- Processing is required at the receiver in order to ensure that the storage data has reached a persistent state, and acknowledge the transfer so that the sender can proceed.

- Typical latency optimizations, such as polling a receive memory location for a key that determines when the data arrives, can create both correctness and security issues because the buffer may not remain stable after the application determines that the IO has completed. This is of particular concern in security conscious environments.

The first issue is fundamental, and due to the nature of serial, shared communication channels, presents challenges that are not easily bypassed. Therefore, an RDMA solution which reduces the exchanges which encounter such latencies is highly desirable.

The second issue requires that outbound transfers be made as

efficient as possible, so that replication of data can be done with minimal overhead and delay (latency). A reliable "push" RDMA transfer method is highly suited to this.

The third issue requires that the transfer be performed without an upper-layer exchange required. Within security contraints, RDMA transfers arbitrated only by lower layers into well-defined and pre-advertised buffers present an ideal solution.

The fourth issue requires significant CPU activity, consuming power and valuable resources, and additionally is not guaranteed by the RDMA protocols, which make no guarantee of the order in which received data is placed or becomes visible; such guarantees are made only after signaling a completion to upper layers.

The RDMAP and DDP protocols, together, provide data transfer semantics with certain consistency guarantees to both the sender and receiver. Delivery of data transferred by these protocols is said to have been Placed in destination buffers upon Completion of specific operations. In general, these guarantees are limited to the visibility of the transferred data within the hardware domain of the receiver (data sink). Significantly, the guarantees do not necessarily extend to the actual storage of the data in memory cells, nor do they convey any guarantee of durability, that is, that the data may not be present after a catastrophic failure such as power

loss. These guarantees may be provided by upper layers, such as the ones mentioned.

The NFSv4.1 and iSER protocols are, respectively, file and block oriented, and have been used extensively for providing access to hard disk and solid state flash drive media. Such devices incur certain latencies in their operation, from the millisecond-order rotational and seek delays of rotating disk hardware, or the 100-microsecond-order erase/write and translation layers of solid state flash. These file and block protocols have benefited from the increased bandwidth, lower latency, and markedly lower CPU overhead of RDMA to provide excellent performance for such media, approximately 30-50 microseconds for 4KB writes in leading implementations.

These protocols employ a "pull" model for write: the client, or

initiator, sends an upper layer write request which contains a reference to the data to be written. The upper layer protocols encode this as one or more memory regions. The server, or target, then prepares the request for local write execution, and "pulls" the data with an RDMA Read. After processing the write, a response is returned. There are therefore two or more roundtrips on the RDMA network in processing the request. This is desirable for several reasons, as described in the relevant specifications, but it incurs latency. However, since as mentioned the network latency has been so much less than the storage processing, this has been a sound approach.

Today, a new class of Storage Class Memory is emerging, in the form of Non-Volatile DIMM and NVM Express devices, among others. These devices are characterized by further reduced latencies, in the 10-microsecond-order range for NVMe, and sub-microsecond for NVDIMM. The 30-50 microsecond write latencies of the above file and block protocols are therefore from one to two orders of magnitude larger than the storage media! The client/server processing model of traditional storage protocols are no longer amortizable at an acceptable level into the overall latency of storage access, due to their requiring request/response communication, CPU processing by the both server and client (or target and initiator), and the interrupts to signal such requests.

Another important property of certain such devices is the requirement for explicitly requesting that the data written to them be made durable. Because durability requires that data be committed to memory cells, it is a relatively expensive operation in time (and power), and in order to maintain the highest device throughput and most efficient operation, the "commit" operation is explicit. When the data is written by an application on the local platform, this responsibility naturally falls to that application (and the CPU on

which it runs). However, when data is written by current RDMA protocols, no such semantic is provided. As a result, upper layer stacks, and the target CPU, must be invoked to perform it, adding overhead and latency that is now highly undesirable.

When such devices are deployed as the remote server, or target, storage, and when such a durability can be requested and guaranteed remotely, a new transfer model can be considered. Instead of relying

on the server, or target, to perform requested processing and to reply after the data is durably stored, it becomes desirable for the client, or initiator, to perform these operations itself. By altering the transfer models to support a "push mode", that is, by allowing the requestor to push data with RDMA Write and subsequently make it durable, a full round trip can be eliminated from the operation. Additionally, the signaling, and processing overheads at the remote peer (server or target) can be eliminated. This becomes an extremely compelling latency advantage.

Together the above discussion argues for a new transfer model supporting remote durability guarantees, provided by the RDMA transport, and used directly by upper layers on a data source, to control durable storage of data on a remote data sink without requiring its remote interaction. Existing, or new, upper layers can use such a model in several ways, and evolutionary steps to support durability guarantees without required protocol changes are explored in the remainder of this document.

Note that is intended that the requirements and concept of these extensions can be applied to any similar RDMA protocol, and that a compatible remote durability model can be applied broadly.

2.1. Requirements

The fundamental new requirement for extending RDMA protocols is to define the property of _durability_. This new property drives the operations to extend Placement as defined in existing RDMA protocols. When Placed, these protocols require only that the data be visible consistently to both the platform on which the buffer resides, and to remote peers across the network via RDMA. In modern hardware designs, this buffer can reside in memory, or also in cache, if that cache is part of the hardware consistency domain. Many designs use such caches extensively to improve performance of local access.

Durability, by contrast, requires that the data not only be consistently visible, it further requires that the buffer contents be preserved across catastrophic failures. While it is possible for caches to be durable, they are typically not. Efficient designs, in fact, lead many implementations to make them volatile. In these

designs, an explicit flush operation, often followed by an explicit

commit, is required to provide this guarantee.

For the RDMA protocol to remotely provide durability guarantees, the new requirement is mandatory. Note that this does not imply support for durability by the RDMA hardware implementation itself; it is entirely acceptable for the RDMA implementation to request durability from another subsystem, for example, by requesting that the CPU perform the flush and commit. But, in an ideal implementation, the RDMA implementation will be able to act as a master and provide these services without further work requests. Note, it is possible that different buffers will require different durability processing, for example one buffer may reside in persistent memory, while another may place its durable blocks in a persistent storage device. Many such memory-addressable designs are entering the market, from NVDIMM to NVMe and even to SSDs and hard drives.

Therefore, any local memory registration primitive will be enhanced to specify an optional durability attribute, along with any local information required to achieve it. These attributes remain local - like existing local memory registration, the region is fully described by a { handle, offset, length } descriptor, and such aspects of the local physical address, memory type, protection (remote read, remote write, protection key), etc are not instantiated in the protocol. The RDMA implementation maintains these, and strictly performs processing based on them, but they are not known to the peer, and therefore are not a matter for the protocol.

Note, additionally, that by describing durability only through the presence of an optional durability attribute, it is possible to describe regions as both durable and non-durable, in order to enable efficient processing. When commit is remotely requested of a non-durable region, the result is not required to be that the data is durable. This can be used by upper layers to enable bulk-type processing with low overhead, by assigning specific durability through use of the Steering Tag.

The intention is that if the underlying region is marked as non-volatile, the placement of data into it is also non-volatile (i.e. any volatile buffering between the network and the underlying storage has been flushed).

To enable the maximum generality, the commit operation is specified to act on a list of { handle, offset, length } regions. The requirement is that each byte of each specified region be made durable before the response to the commit is generated. However, depending on the implementation, other bytes in other regions may be made durable as part of processing any commit. Any data in any

buffer destined for persistent, durable storage, may become durable at any time, even if not requested explicitly. For example, a simple and stateless approach would be for all data be flushed and committed, system-wide. A possibly more efficient implementation might track previously written bytes, or blocks with "dirty" bytes, and commit only those. Either result provides the required guarantee. The length of the region list, and the maximum amount of data that can be made durable in a single request, are implementation dependent and its protocol expression is to be described.

The commit operation is specified to return a status, which may be zero on success but may take other values to be determined. Several possibilities present themselves. The commit operation may fail to make the data durable, perhaps due to a hardware failure, or a change in device capability (device read-only, device wear, etc). The data, however, may not have been lost and is still present in the buffer. Or, the device may support an integrity check, similar to modern error checking memory or media error detection on hard drive surfaces, and its status is returned. Or, the request may exceed device limits in size or even transient attribute such as temporary media failure. The behavior of the device itself is beyond the scope of this specification.

Because the commit involves processing on the local platform and the actual device, it is expected to take a certain time to be performed. For this reason, the commit operation is required to be defined as a "queued" operation on the RDMA device, and therefore also the protocol. The RDMA protocol supports RDMA Read and Atomic in such a fashion. The iWARP family defines a "queue number" with queue-specific processing that is naturally suited for this. Queuing provides a convenient means for supporting ordering among other operations, and for flow control. Flow control for RDMA Reads and Atomics share incoming and outgoing crediting depths ("IRD/ORD"); commit will either share these, or define their own separate values.

2.1.1. Non-Requirements

The protocol does not include a "RDMA Write with durability", that is, a modifier on the existing RDMA Write operation. While it might seem a logical approach, several issues become apparent:

The existing RDMA Write operation is unacknowledged at the RDMA layer. Requiring it to provide an indication of remote durability would require it to have an acknowledgement, which would be an undesirable extension to the operation.

Such an operation would require flow control and therefore also buffering on the responding peer. Existing RDMA Write semantics

are not flow controlled and as tagged transfers are by design zero-copy i.e. unbuffered. Requiring these would introduce potential pipeline stalls and increase implementation complexity in a critical performance path.

The operation at the initiator would stall until the acknowledgement of completion, significantly changing the semantic of the existing operation, and complicating software by blocking the send work queue. As each operation would be self-describing with respect to durability, individual operations would therefore block with differing semantics.

Even for the possibly-common case of commiting after every write, it is highly undesirable to impose new optional semantics on an existing operation. And, the same result can be achieved by sending the commit in the same network packet, and since the RDMA Write is unacknowledged while the commit is always replied-to, no additional overhead is imposed on the combined exchange.

[Further expand on the undesirable nature of such a change.]

2.2. Additional Semantics

Ordering w.r.t. RDMA Write, receives, RDMA Read, other commits. Also, ensure ordering ensures similar remote semantics to local

The commit operation is ordered with respect to certain other operations, and it may be advantageous to combine certain actions into the same request, or requests with specific ordering to the commit. Examples to be discussed include:

Additional optional payload to be placed and made durable in an atomic fashion after the requested commit. A small (64 bit) payload, sent in the same, or other single, request, and aligned such that it can be made durable in a single hardware operation, can be used to satisfy the "log update" scenario (describe this in more detail).

Immediate data to be optionally provided in a completion to an

upper layer on the remote peer. Such an indication can be used to signal the upper layer that certain data has been placed in the peer's buffer, and has been made available durably.

Remote invalidation, as optionally performed by existing RDMA protocols for other operations.

Upper Layer message, an optional full message to be provided in a completion after the commit.

Integrity check for committed data, which could take the form of a value to be verified before returning, or a value computed and returned which the initiator can use to verify. Specification of the checksum or hash algorithm, or its negotiation by an upper layer, will be necessary if adopted.

3. Proposed Extensions

The extensions in this document fall into two categories:

- Local behavior extensions
- Protocol extensions

These categories are described, and may be implemented, separately.

3.1. Local Extensions

Here discuss memory registration, new memory and protection attributes, and applicability to both remote and "local" (receives).

3.1.1. Registration Semantics

New platform-specific attributes to RDMA registration, allows them to be processed at the server *only* without client knowledge, or protocol exposure. No client knowledge - ensures future interop

New local PM memory registration example:

Register(region[], PMType, mode) -> Handle

PMType includes type of PM i.e. plain RAM, or "commit required", or PCIe-resident, or any other local platform-specific processing

Mode includes disposition of data Read and/or write e.g. Cacheable after operation (needed by CPU on data sink)

Handle is processed in receiving NIC during RDMA operation to specified region, under control of original Mode.

Also consider whether potential "integrity check" behavior can be made per-region. If so, piggybacking it on the registration enables selecting the integrity hash, and making its processing optional and straightforward.

Any other per-region durability processing to be explored.

3.1.2. Completion Semantics

Transparency is possible when upper layer provides Completions (e.g. messages or immediate data)

Commit to durability can be piggybacked by data sink upon signaling. Upper layer may not need to explicitly Commit in this case, which of course are dependent on upper layer and workload.

Can apply this concept to RDMA Write with Immediate Or ...ordinary receives. Strong possibilities exist - explore here.

Ordering of operations is critical: Such RDMA Writes cannot be allowed to "pass" durability. Therefore, protocol implications may also exist.

Discuss optional behaviors explored in prior section, and whether/how they generate completions.

3.1.3. Platform Semantics

Writethrough behavior on durable regions and reasons for same. Consider requiring/recommending a local writethrough behavior on any

durable region, to support a nonblocking hurry-up to avoid future stalls on a subsequent cache flush, prior to a commit. Also, it would enhance durability.

PCI extension to support Commit Allow NIC to provide durability directly and efficiently To Memory, CPU, PCI Root, PM device, PCIe device, ... Avoids CPU interaction Supports strong data consistency model Performs equivalent of: CLFLUSHOPT (region list) PCOMMIT Or if NIC is on memory bus or within CPU complex... Other possibilities exist

3.2. RDMAP Extensions

This document defines a new RDMA operation, "RDMA Commit". The wire-related aspects of the proposed protocol are discussed in this section.

This section and the ones following present one possible approach toward defining the wire protocol defined by the above discussion. The definitions are included for initial discussion and do not comprise a complete specification. Certain additional protocol features of any potential new extension, such as any associated

Immediate Data, Solicited Events, Remote Invalidation, ULP Message inclusion, etc, are left to a later version.

For reference, Figure 1 depicts the format of the DDP Control and RDMAP Control Fields, in the style and convention of RFC 5040 and RFC7306:

The DDP Control Field consists of the T (Tagged), L (Last), Resrv, and DV (DDP protocol Version) fields RFC 5041. The RDMAP Control Field consists of the RV (RDMA Version), Rsv, and Opcode fields RFC 5040.

This specification adds values for the RDMA Opcode field to those specified in RFC 5040. Table 1 defines the new values of the RDMA Opcode field that are used for the RDMA Messages defined in this specification.

As shown in Table 1, STag (Steering Tag) and Tagged Offset are valid

only for certain RDMA Messages defined in this specification. Table 1 also shows the appropriate Queue Number for each Opcode.

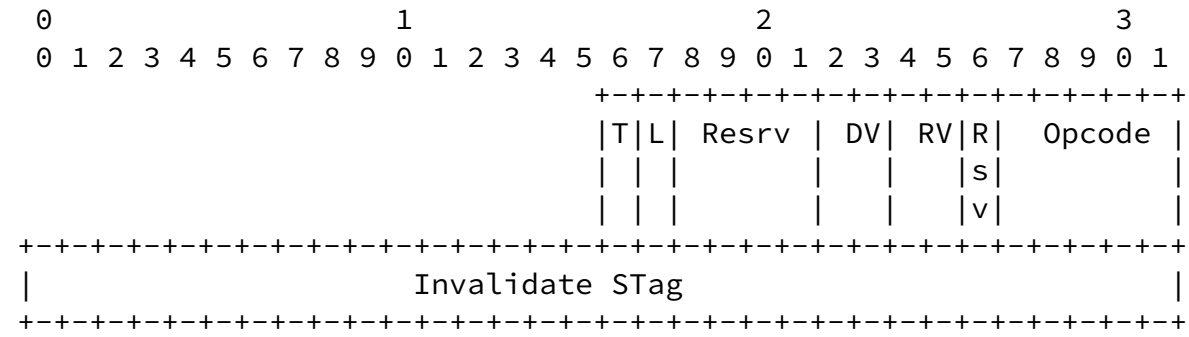

Figure 1: DDP Control and RDMAP Control Fields

All RDMA Messages defined in this specification MUST carry the following values:

- The RDMA Version (RV) field: 01b.
- Opcode field: Set to one of the values in Table 1.
- Invalidate STag: Set to zero, or optionally to non-zero by the sender, processed by the receiver.

Note: N/A in the table below means Not Applicable

RDMA Opcode	Message Type	Tagged Flag	STag and TO	Queue Number	Invalidate STag	Message Length Communicated between DDP and RDMAP
01100b	RDMA Commit Request	0	N/A	1	opt	Yes

01101b	RDMA Commit Response	0	N/A	3	N/A	Yes

Table 1: Additional RDMA Usage of DDP Fields

This extension adds RDMAP use of Queue Number 1 for Untagged Buffers for issuing RDMA Commit Requests, and use of Queue Number 3 for Untagged Buffers for tracking RDMA Commit Responses.

All other DDP and RDMAP Control Fields are set as described in RFC 5040 and RFC 7306.

Table 2 defines which RDMA Headers are used on each new RDMA Message and which new RDMA Messages are allowed to carry ULP payload.

RDMA Message OpCode	Message Type	RDMA Header Used	ULP Message allowed in the RDMA Message
01100b	RDMA Commit Request	None	TBD
01101b	RDMA Commit Response	None	No

Table 2: RDMA Message Definitions

Further discussion.

3.2.1. RDMA Commit Request Header Format

The RDMA Commit Request Message makes use of the DDP Untagged Buffer Model. RDMA Commit Request messages MUST use the same Queue Number

as RDMA Read Requests and RDMA Extensions Atomic Operation Requests (QN=1). Reusing the same queue number for RMDA Commit Requests allows the operations to reuse the same infrastructure (e.g. Outbound and Inbound RDMA Read Queue Depth (ORD/IRD) flow control) as that defined for RDMA Read Requests.

The RDMA Commit Request Message carries an RDMA Commit header that describes the ULP Buffer address in the Responder's memory. The RDMA Write Request header immediately follows the DDP header. The RDMAP layer passes an RDMAP Control Field to the DDP layer. Figure 2 depicts the RDMA Commit Request Header that is used for all RDMA Commit Request Messages:

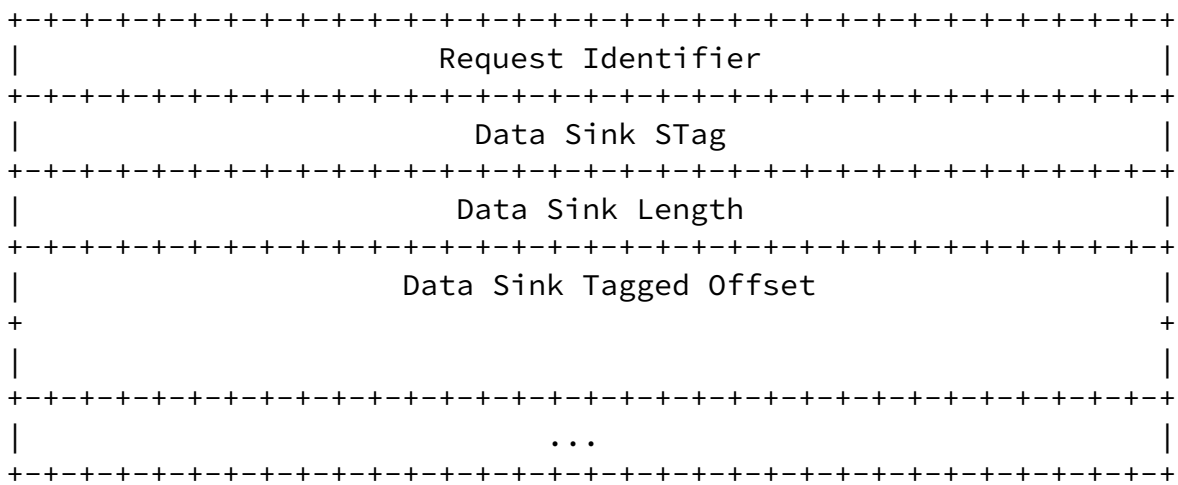

Figure 2: RDMA Commit Request Header

Request Identifier: 32 bits. The Request Identifier specifies a number that is used to identify the RDMA Commit Request Message. The value used in this field is selected by the RNIC that sends the message, and it is reflected back to the Local Peer in the RDMA Commit Response message. N.B. Is this field really useful to the RNIC, or does ordering suffice???

Data Sink STag: 32 bits The Data Sink STag identifies the Remote Peer's Tagged Buffer targeted by the RDMA Commit Request. The Data Sink STag is associated with the RDMAP Stream through a mechanism that is outside the scope of the RDMAP specification.

Data Sink Tagged Offset: 64 bits The Data Sink Tagged Offset specifies the starting offset, in octets, from the base of the Remote Peer's Tagged Buffer targeted by the RDMA Commit Request.

... Additional region identifiers to be committed in processing the RDMA Commit Request, and/or upper layer message to be passed to upper layer after commit completion (TBD).

3.2.2. RDMA Commit Response Header Format

The RDMA Commit Response Message makes use of the DDP Untagged Buffer Model. RDMA Commit Response messages MUST use the same Queue Number as RDMA Extensions Atomic Operation Responses (QN=3). The RDMAP layer passes the following payload to the DDP layer on Queue Number 3.

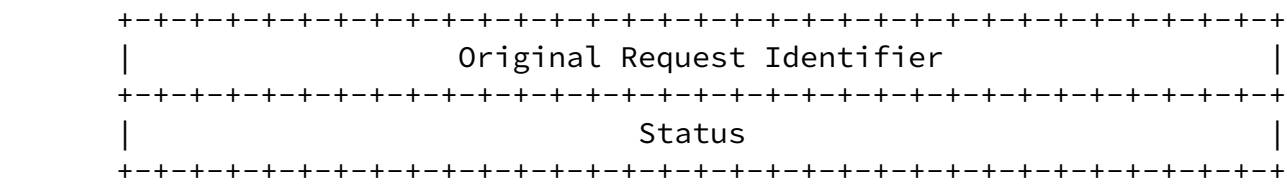

Figure 3: RDMA Commit Response Header

Original Request Identifier: 32 bits. The Original Request Identifier is set to the value specified in the Request Identifier field that was originally provided in the corresponding RDMA Commit Request Message. N.B. ditto previous question.

Status: 32 bits. Zero if the RDMA Commit was successfully processed, or any other value if not.

3.2.3. Ordering

Ordering and completion rules for RDMA Commit Request are similar to those for an Atomic operation as described in section 5 of RFC 7306. The queue number field of the RDMA Commit Request for the DDP layer MUST be 1, and the RDMA Commit Response for the DDP layer MUST be 3.

There are no ordering requirements for the placement of the data to be committed, nor are there any requirements for the order in which the data is made durable. Data received by prior operations (e.g. RDMA Write) MAY be submitted for placement at any time, and durability MAY occur before the commit is requested. Data committed after placement MAY become durable at any time, in the course of operation of the persistency management of the storage device, or by other actions resulting in durability. Any data specified by the commit operation, in any case, MUST be made durable before successful return of the commit.

3.2.4. Atomicity

There are no atomicity guarantees provided on the Responder's node by the RDMA Commit Operation with respect to any other operations. While the Completion of the RDMA Commit Operation ensures that the requested data was placed and committed to the target Tagged Buffer, other operations might have also placed or fetched overlapping data. The upper layer is responsible for arbitrating any shared access.

(To discuss) The commit operation provides an optional block of data which is committed to a specified region after the successful completion of the requested commit. This specified region MAY be constrained in size and alignment by the implementation, and the implementation MUST fail the operation and send a terminate message if the subsequent commit cannot be performed atomically. The implementation MUST NOT perform the subsequent commit if an error occurred on the requested commit, and SHOULD return a non-zero status indicating the error.

(Sidebar) It would be useful to make a statement about other RDMA Commit to the target buffer and RDMA Read from the target buffer on the same connection. Use of QN 1 for these operations provides ordering guarantees which imply that they will "work" (see #7 below). NOTE: this does not, however, extend to RDMA Write, which is not sequenced nor does it employ a DDP QN.

3.2.5. Discovery of RDMAP Extensions

As for RFC 7306, explicit negotiation by the RDMAP peers of the extensions covered by this document is not required. Instead, it is RECOMMENDED that RDMA applications and/or ULPs negotiate any use of these extensions at the application or ULP level. The definition of such application-specific mechanisms is outside the scope of this specification. For backward compatibility, existing applications and/or ULPs should not assume that these extensions are supported.

In the absence of application-specific negotiation of the features defined within this specification, the new operations can be attempted, and reported errors can be used to determine a remote peer's capabilities. In the case of RDMA Commit, an operation to a

previously Advertised buffer with remote write permission can be used to determine the peer's support. A Remote Operation Error or Unexpected OpCode error will be reported by the remote peer if the Operation is not supported by the remote peer.

4. Ordering and Completions Table

Table 3 summarizes the ordering relationships for the RDMA Commit operation from the standpoint of the Requester. Note that in the table, Send Operation includes Send, Send with Invalidate, Send with Solicited Event, and Send with Solicited Event and Invalidate. Also note that Immediate Operation includes Immediate Data and Immediate Data with Solicited Event.

As for the prior section, the text below presents one possible approach, and is included in skeletal form to be filled-in when appropriate.

Note: N/A in the table below means Not Applicable

First Operation	Second Operation	Placement Guarantee at Remote Peer	Placement Guarantee at Local Peer	Ordering Guarantee at Remote Peer
RDMA Commit	TODO	No Placement Guarantee between Foo and Bar	N/A	Completed in Order
TODO	RDMA Commit	No Placement Guarantee between Foo and Bar	N/A	TODO
TODO	TODO	Etc	Etc	Etc

Table 3: Ordering of Operations

5. Error Processing

In addition to error processing described in section 7 of RFC 5040 and section 8 of RFC 7306, the following rules apply for the new RDMA Messages defined in this specification.

5.1. Errors Detected at the Local Peer

The Local Peer MUST send a Terminate Message for each of the following cases:

1. For errors detected while creating a RDMA Commit Request or other reasons not directly associated with an incoming Message, the Terminate Message and Error code are sent instead of the Message. In this case, the Error Type and Error Code fields are included in the Terminate Message, but the Terminated DDP Header and Terminated RDMA Header fields are set to zero.

2. For errors detected on an incoming RDMA Commit Request or RDMA Commit Response, the Terminate Message is sent at the earliest possible opportunity, preferably in the next outgoing RDMA Message. In this case, the Error Type, Error Code, and Terminated DDP Header fields are included in the Terminate Message, but the Terminated RDMA Header field is set to zero.

3. For errors detected in the processing of the RDMA Commit itself, that is, the act of making the data durable, no Terminate Message is generated. Because the data is not lost, the connection MUST NOT terminate and the peer MUST inform the requester of the status, and allow the requester to perform further action, for instance, recovery.

5.2. Errors Detected at the Remote Peer

On incoming RDMA Commit Requests, the following MUST be validated:

- The DDP layer MUST validate all DDP Segment fields.

The following additional validation MUST be performed:

- If the RDMA Commit cannot be satisfied, due to transient or permanent errors detected in the processing by the Responder, a status MUST be returned to the Requestor. Valid status values are to be specified.

6. IANA Considerations

This document requests that IANA assign the following new operation codes in the "RDMAP Message Operation Codes" registry defined in section 3.4 of [RFC6580].

0xC RDMA Commit Request, this specification

0xD RDMA Commit Response, this specification

Additionally, the name of the listed entry in "RDMAP DDP Untagged Queue Numbers" as defined in section 10.2 of [RFC7306] is requested to be updated as follows:

0x00000003 Queue 3 Modify name to "Atomic Response and RDMA Commit Response operations" and add reference to this specification

Note to RFC Editor: this section may be edited and updated prior to publication as an RFC.

7. Security Considerations

This document specifies extensions to the RDMA Protocol specification in RFC 5040 and RDMA Protocol Extensions in RFC 7306, and as such the Security Considerations discussed in Section 8 of RFC 5040 and Section 9 of RFC 7306 apply. In particular, RDMA Commit Operations use ULP Buffer addresses for the Remote Peer Buffer addressing used in RFC 5040 as required by the security model described in [RDMAP Security [RFC5042]].

If the "push mode" transfer model discussed in section 2 is implemented by upper layers, new security considerations will be potentially introduced in those protocols, particularly on the

server, or target, if the new memory regions are not carefully protected. Therefore, for them to take full advantage of the extension defined in this document, additional security design is required in the implementation of those upper layers. The facilities of RFC5042 [RFC5042] can provide the basis for any such design.

In addition to protection, in "push mode" the server or target will expose memory resources to the peer for potentially extended periods, and will allow the peer to perform remote durability requests which will necessarily consume shared resources, e.g. memory bandwidth, power, and memory itself. It is recommended that the upper layers provide a means to gracefully adjust such resources, for example using upper layer callbacks, without resorting to revoking RDMA permissions, which would summarily close connections. With the initiator applications relying on the protocol extension itself for managing their required durability, the lack of such an approach would lead to frequent recovery in low-resource situations, potentially opening a new threat to such applications.

8. References

8.1. Normative References

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

[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>.

[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>.

[RFC6580] Ko, M. and D. Black, "IANA Registries for the Remote Direct Data Placement (RDDP) Protocols", RFC 6580, DOI 10.17487/RFC6580, April 2012, <http://www.rfc-editor.org/info/rfc6580>.

[RFC7306] Shah, H., Marti, F., Noureddine, W., Eiriksson, A., and R. Sharp, "Remote Direct Memory Access (RDMA) Protocol Extensions", RFC 7306, DOI 10.17487/RFC7306, June 2014, <http://www.rfc-editor.org/info/rfc7306>.

8.2. Informative References

[RFC5045] Bestler, C., Ed. and L. Coene, "Applicability of Remote Direct Memory Access Protocol (RDMA) and Direct Data Placement (DDP)", RFC 5045, DOI 10.17487/RFC5045, October 2007, <http://www.rfc-editor.org/info/rfc5045>.

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

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

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

[RFC7145] Ko, M. and A. Nezhinsky, "Internet Small Computer System Interface (iSCSI) Extensions for the Remote Direct Memory Access (RDMA) Specification", RFC 7145, DOI 10.17487/RFC7145, April 2014, <http://www.rfc-editor.org/info/rfc7145>.

[SCSI] American National Standards Institute, "SCSI Primary Commands - 3 (SPC-3) (INCITS 408-2005)", May 2005.

[SMB3] Microsoft Corporation, "Server Message Block (SMB) Protocol Versions 2 and 3 (MS-SMB2)", October 2015.

[SMBDirect] Microsoft Corporation, "SMB2 Remote Direct Memory Access (RDMA) Transport Protocol (MS-SMBD)", October 2015.

[SNIANVM] Storage Networking Industry Association NVM TWG, "SNIA NVM Programming Model v1.0", 2014.

8.3. URIs

[1] http://www.nvmexpress.org

[2] http://www.jedec.org

Appendix A. DDP Segment Formats for RDMA Extensions

This appendix is for information only and is NOT part of the standard. It simply depicts the DDP Segment format for each of the RDMA Messages defined in this specification.

A.1. DDP Segment for RDMA Commit Request

Figure 3 depicts an RDMA Commit Request, DDP Segment:

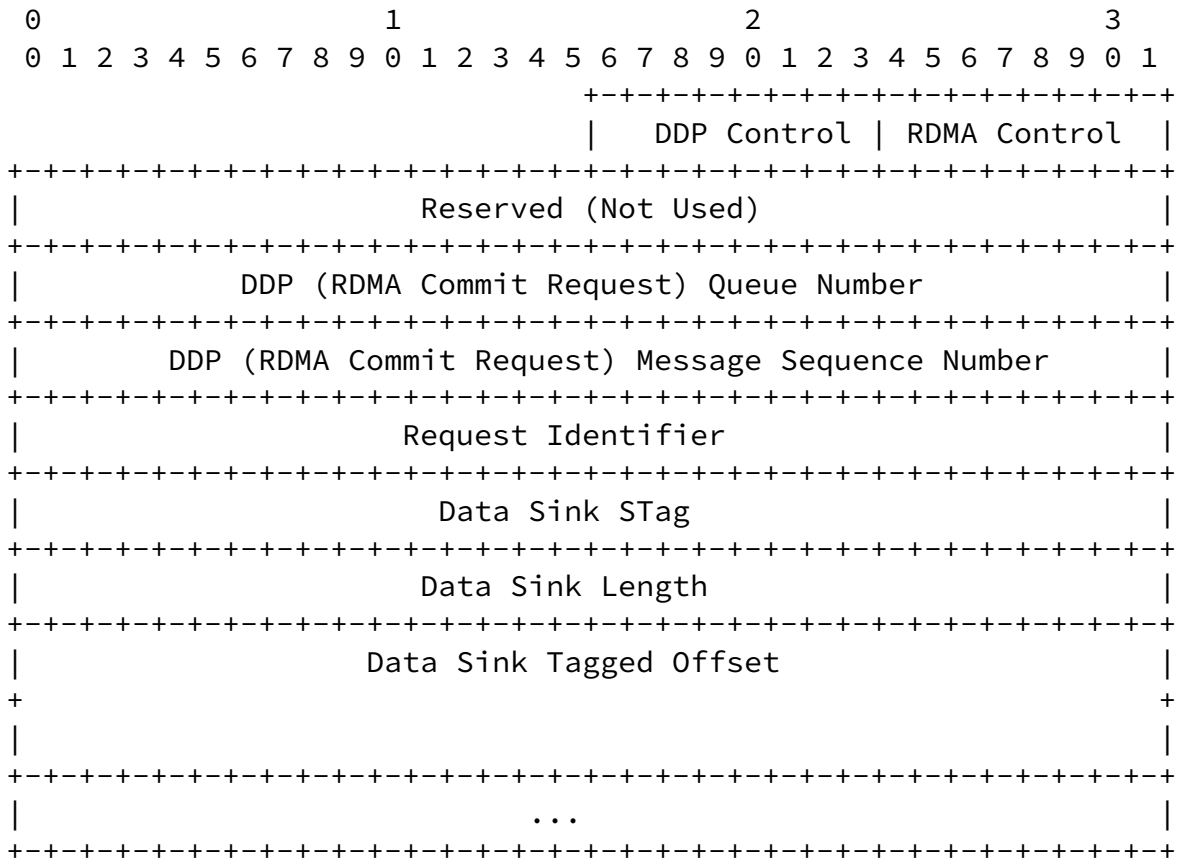

Figure 3

A.2. DDP Segment for RDMA Commit Response

Figure 4 depicts an RDMA Commit Response, DDP Segment:

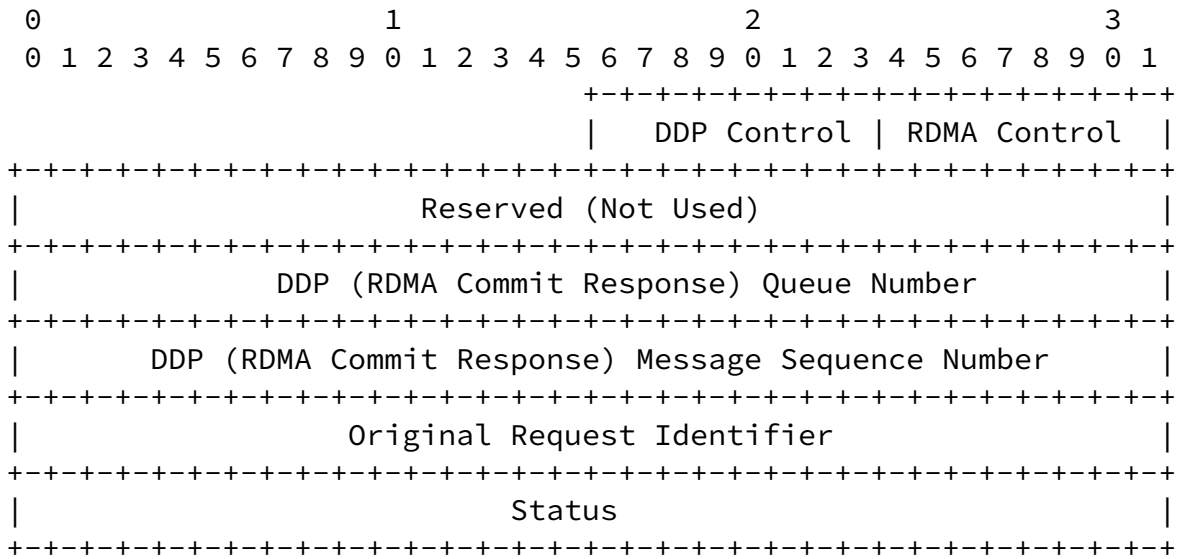

Figure 4

Authors' Addresses

Tom Talpey
Microsoft
One Microsoft Way
Redmond, WA 98052
US

Email: ttalpey@microsoft.com

Jim Pinkerton
Microsoft
One Microsoft Way
Redmond, WA 98052
US

Email: jpink@microsoft.com