NFSv4 (provisionally)
Internet-Draft
Updates: 5040 7306 (if approved)
Intended status: Standards Track
Expires: September 10, 2020

T. Talpey
Microsoft
T. Hurson
Intel
G. Agarwal
Marvell
T. Reu
Chelsio

March 9, 2020

RDMA Extensions for Enhanced Memory Placement
draft-talpey-rdma-commit-01

Abstract

This document specifies extensions to RDMA (Remote Direct Memory Access) protocols to provide capabilities in support of enhanced remotely-directed data placement on persistent memory-addressable devices. The extensions include new operations supporting remote commitment to persistence 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 updates RFC5040 (Remote Direct Memory Access Protocol (RDMAP)) and updates RFC7306 (RDMA Protocol Extensions).

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 https://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 September 10, 2020.

Copyright Notice

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 SNIANVMP [SNIANVMP] 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 persistence and/or visibility without invoking upper layers and remote CPUs for each such request, are the motivators for the extensions in this document.

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

- Flush - support for RDMA requests and responses with enhanced placement semantics.
- Atomic Write - support for writing certain data elements into memory in an atomically visible fashion.
- Verify - support for validating the contents of remote memory, through use of integrity signatures.
- Enhanced memory registration semantics in support of persistence and visibility.

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 RFC7306, and key words are additionally defined in the glossaries of the referenced documents.

The following additional terms are used in this document as defined.

Flush: The submitting of previously written data from volatile intermediate locations for subsequent placement, in a persistent and/or globally visible fashion.

Invalidate: The removal of data from volatile intermediate locations.

Commit: Obsolescent previous synonym for Flush. Term to be deleted.

Persistent: The property that data is present, readable 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].

Globally Visible: The property of data being available for reading consistently by all processing elements on a system. Global visibility and persistence are not necessarily causally related; either one may precede the other, or they may take effect simultaneously, depending on the architecture of the platform.

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 100 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 average 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 increasing 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] RFC8166 [RFC8166] RFC8267 [RFC8267], 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 highly 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 this approach requires the memory remain open to writes and therefore 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. Communication cannot exceed the speed of light, for example, and serialization/deserialization plus packet processing adds further delay. Therefore, an RDMA solution which offloads and

reduces the overhead of 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 may not be guaranteed by the RDMA protocols, which make no requirement of the order in which certain 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 that the data integrity is intact, nor that it remains present after a catastrophic failure. These guarantees may be provided by upper layers, such as the ones mentioned, after processing the Completions, and performing the necessary operations.

The NFSv4.1, SMB3 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 an RDMA 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 persistent. Because persistence 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 device "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 persistence 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 persistently 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 persistent, 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.

In DDP (RFC5041), data is considered "placed" when it is submitted by the RNIC to the system. This operation is commonly an i/o bus write, e.g. via PCI. The submission is ordered, but there is no confirmation or necessary guarantee that the data has yet reached its destination, nor become visible to other devices in the system. The data will eventually do so, but possibly at a later time. The act of "delivery", on the other hand, offers a stronger semantic, guaranteeing that not only have prior operations been executed, but also guaranteeing any data is in a consistent and visible state. Generally however, such "delivery" requires raising a completion event, necessarily involving the host CPU. This is a relatively expensive, and latency-bound operation. Some systems perform "DMA snooping" to provide a somewhat higher guarantee of visibility after delivery and without CPU intervention, but others do not. The RDMA requirements remain the same, therefore, upper layers may make no broad assumption. Such platform behaviors, in any case, do not address persistence.

The extensions in this document primarily address a new "flush to persistence" RDMA operation. This operation, when invoked by a connected remote RDMA peer, can be used to request that previously-written data be moved into the persistent storage domain. This may be a simple flush to a memory cell, or it may require movement across one or more busses within the target platform, followed by an explicit persistence operation. Such matters are beyond the scope of this specification, which provides only the mechanism to request the operation, and to signal its successful completion.

In a similar vein, many applications desire to achieve visibility of remotely-provided data, and to do so with minimum latency. One example of such applications is "network shared memory", where publish-subscribe access to network-accessible buffers is shared by multiple peers, possibly from applications on the platform hosting the buffers, and others via network connection. There may therefore be multiple local devices accessing the buffer - for example, CPUs, and other RNICs. The topology of the hosting platform may be complex, with multiple i/o, memory, and interconnect busses, requiring multiple intervening steps to process arriving data.

To address this, the extension additionally provides a "flush to global visibility", which requires the RNIC to perform platform-dependent processing in order to guarantee that the contents of a specific range are visible for all devices that access them. On certain highly-consistent platforms, this may be provided natively. On others, it may require platform-specific processing, to flush data from volatile caches, invalidate stale cached data from others, and to empty queued pending operations. Ideally, but not universally, this processing will take place without CPU intervention. With a

global visibility guarantee, network shared memory and similar applications will be assured of broader compatibility and lower latency across all hardware platforms.

Subsequently, many applications will seek to obtain a guarantee that the integrity of the data has been preserved after it has been flushed to a persistent or globally visible state. This may be enforced at any time. Unlike traditional block-based storage, the data provided by RDMA is neither structured nor segmented, and is therefore not self-describing with respect to integrity. Only the originator of the data, or an upper layer, is in possession of that. Applications requiring such guarantees may include filesystem or database logwriters, replication agents, etc.

To provide an additional integrity guarantee, a new operation is provided by the extension, which will calculate, and optionally compare an integrity value for an arbitrary region. The operation is ordered with respect to preceding and subsequent operations, allowing for a request pipeline without "bubbles" - roundtrip delays to ascertain success or failure.

Finally, once data has been transmitted and directly placed by RDMA, flushed to its final state, and its integrity verified, applications will seek to commit the result with a transaction semantic. The previous application examples apply here, logwriters and replication are key, and both are highly latency- and integrity-sensitive. They desire a pipelined transaction marker which is placed atomically to indicate the validity of the preceding operations. They may require that the data be in a persistent and/or globally visibile state, before placing this marker.

Together the above discussion argues for a new "one sided" transfer model supporting extended remote placement guarantees, provided by the RDMA transport, and used directly by upper layers on a data source, to control persistent 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 persistence 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 model can be applied broadly.

2.1. Requirements for RDMA Flush

The fundamental new requirement for extending RDMA protocols is to define the property of _persistence_. This new property is to be expressed by new operations to extend Placement as defined in existing RDMA protocols. The RFC5040 protocols specify that Placement means that the data is visible consistently within a platform-defined domain on which the buffer resides, and to remote peers across the network via RDMA to an adapter within the domain. 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.

Persistence, by contrast, requires that the buffer contents be preserved across catastrophic failures. While it is possible for caches to be persistent, they are typically not, or they provide the persistence guarantee for a limited period of time, for example, while backup power is applied. Efficient designs, in fact, lead most implementations to simply make them volatile. In these designs, an explicit flush operation (writing dirty data from caches), often followed by an explicit commit (ensuring the data has reached its destination and is in a persistent state), is required to provide this guarantee. In some platforms, these operations may be combined.

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

Therefore, additionally any local memory registration primitive will be enhanced to specify new optional placement attributes, along with any local information required to achieve them. These attributes do not explicitly traverse the network - like existing local memory registration, the region is fully described by a { STag, Tagged 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. Indeed,

each local RDMA implementation maintains these, and strictly performs processing based on them, and they are not exposed to the peer. Such considerations are discussed in the security model [RDMAP Security [RFC5042]].

Note, additionally, that by describing such attributes only through the presence of an optional property of each region, it is possible to describe regions referring to the same physical segment as a combination of attributes, in order to enable efficient processing. Processing of writes to regions marked as persistent, globally visible, or neither ("ordinary" memory) may be optimized appropriately. For example, such memory can be registered multiple times, yielding multiple different Steering Tags which nonetheless merge data in the underlying memory. This can be used by upper layers to enable bulk-type processing with low overhead, by assigning specific attributes through use of the Steering Tag.

When the underlying region is marked as persistent, that the placement of data into persistence is guaranteed only after a successful RDMA Flush directed to the Steering Tag which holds the persistent attribute (i.e. any volatile buffering between the network and the underlying storage has been flushed, and the appropriate platform- and device-specific steps have been performed).

To enable the maximum generality, the RDMA Flush operation is specified to act on a set of bytes in a region, specified by a standard RDMA { STag, Tagged offset, length } descriptor. It is required that each byte of the specified segment be in the requested state before the response to the Flush is generated. However, depending on the implementation, other bytes in the region, or in other regions, may be acted upon as part of processing any RDMA Flush. In fact, any data in any buffer destined for persistent storage, may become persistent at any time, even if not requested explicitly. For example, the host system may flush cache entries due to cache pressure, or as part of platform housekeeping activities. Or, a simple and stateless approach to flushing a specific range might be for all data be flushed and made persistent, system-wide. A possibly more efficient implementation might track previously written bytes, or blocks with "dirty" bytes, and flush only those to persistence. Either result provides the required guarantee.

The RDMA Flush operation provides a response but does not return a status, or can result in an RDMA Terminate event upon failure. A region permission check is performed first, and may fail prior to any attempt to process data. The RDMA Flush operation may fail to make the data persistent, perhaps due to a hardware failure, or a change in device capability (device read-only, device wear, etc). The device itself may support an integrity check, similar to modern error

checking and corection (ECC) memory or media error detection on hard drive surfaces, which may signal failure. 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 RDMA Flush involves processing on the local platform and the actual storage device, in addition to being ordered with certain other RDMA operations, it is expected to take a certain time to be performed. For this reason, the 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 (RFC5040) and Atomic (RFC7306) 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 on any given Queue Pair share incoming and outgoing crediting depths ("IRD/ORD"); operations in this specification share these values and do not define their own separate values.

2.1.1. Non-Requirements

The extension does not include a "RDMA Write to persistence", 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 a tagged DDP request which is unacknowledged at the DDP layer (RFC5042). Requiring it to provide an indication of remote persistence would require it to have an acknowledgement, which would be an undesirable extension to the existing defined 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 requesting peer would stall until the acknowledgement of completion, significantly changing the semantic of the existing operation, and complicating software by blocking the send work queue, a significant new semantic for RDMA Write work requests. As each operation would be self-describing with respect to persistence, individual operations would therefore block with differing semantics and complicate the situation even further.

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

For these reasons, it is deemed a non-requirement to extend the existing RDMA Write operation.

Similarly, the extension does not consider the use of RDMA Read to implement Flush. Historically, an RDMA Read has been used by applications to ensure that previously written data has been processed by the responding RNIC and has been submitted for ordered Placement. However, this is inadequate for implementing the required RDMA Flush:

> RDMA Read guarantees only that previously written data has been Placed, it provides no such guarantee that the data has reached its destination buffer. In practice, an RNIC satisfies the RDMA Read requirement by simply issuing all PCIe Writes prior to issuing any PCIe Reads.
>
> Such PCIe Reads must be issued by the RNIC after all such PCIe Writes, therefore flushing a large region requires the RNIC and its attached bus to strictly order (and not cache) its writes, to "scoreboard" its writes, or to perform PCIe Reads to the entire region. The former approach is significantly complex and expensive, and the latter approach requires a large amount of PCIe and network read bandwidth, which are often unnecessary and expensive. The Reads, in any event, may be satisfied by platform-specfic caches, never actually reaching the destination memory or other device.
>
> The RDMA Read may begin execution at any time once the request is fully received, queued, and the prior RDMA Write requirement has been satisfied. This means that the RDMA Read operation may not be ordered with respect to other queued operations, such as Verify and Atomic Write, in addition to other RDMA Flush operations.
>
> The RDMA Read has no specific error semantic to detect failure, and the response may be generated from any cached data in a consistently Placed state, regardless of where it may reside. For this reason, an RDMA Read may proceed without necessarily verifying that a previously ordered "flush" has succeeded or failed.

RDMA Read is heavily used by existing RDMA consumers, and the semantics are therefore implemented by the existing specification. For new applications to further expect an extended RDMA Read behavior would require an upper layer negotiation to determine if the data sink platform and RNIC appropriately implemented them, or to silently ignore the requirement, with the resulting failure to meet the requirement. An explicit extension, rather than depending on an overloaded side effect, ensures this will not occur.

Again, for these reasons, it is deemed a non-requirement to reuse or extend the existing RDMA Read operation.

Therefore, no changes to existing specified RDMA operations are proposed, and the protocol is unchanged if the extensions are not invoked.

2.2. Requirements for Atomic Write

The persistence of data is a key property by which applications implement transactional behavior. Transactional applications, such as databases and log-based filesystems, among many others, implement a "two phase commit" wherein a write is made durable, and *only upon success*, a validity indicator for the written data is set. Such semantics are challenging to provide over an RDMA fabric, as it exists today. The RDMA Write operation does not generate an acknowledgement at the RDMA layers. And, even when an RDMA Write is delivered, if the destination region is persistent, its data can be made persistent at any time, even before a Flush is requested. Out-of-order DDP processing, packet fragmentation, and other matters of scheduling transfers can introduce partial delivery and ordering differences. If a region is made persistent, or even globally visible, before such sequences are complete, significant application-layer inconsistencies can result. Therefore, applications may require fine-grained control over the placement of bytes. In current RDMA storage solutions, these semantics are implemented in upper layers, potentially with additional upper layer message signaling, and corresponding roundtrips and blocking behaviors.

In addition to controlling placement of bytes, the ordering of such placement can be important. By providing an ordered relationship among write and flush operations, a basic transaction scenario can be constructed, in a way which can function with equal semantics both locally and remotely. In a "log-based" scenario, for example, a relatively large segment (log "record") is placed, and made durable. Once persistence of the segment is assured, a second small segment (log "pointer") is written, and optionally also made persistent. The visibility of the second segment is used to imply the validity, and

persistence, of the first. Any sequence of such log-operation pairs can thereby always have a single valid state. In case of failure, the resulting string (log) of transactions can therefore be recovered up to and including the final state.

Such semantics are typically a challenge to implement on general purpose hardware platforms, and a variety of application approaches have become common. Generally, they employ a small, well-aligned atom of storage for the second segment (the one used for validity). For example, an integer or pointer, aligned to natural memory address boundaries and CPU and other cache attributes, is stored using instructions which provide for atomic placement. Existing RDMA protocols, however, provide no such capability.

This document specifies an Atomic Write extension, which, appropriately constrained, can serve to provide similar semantics. A small (64 bit) payload, sent in a request which is ordered with respect to prior RDMA Flush operations on the same stream and targeted at a segment which is aligned such that it can be placed in a single hardware operation, can be used to satisfy the previously described scenario. Note that the visibility of this payload can also serve as an indication that all prior operations have succeeded, enabling a highly efficient application-visible memory semaphore.

2.3. Requirements for RDMA Verify

An additional matter remains with persistence - the integrity of the persistent data. Typically, storage stacks such as filesystems and media approches such as SCSI T10 DIF or filesystem integrity checks such as ZFS provide for block- oir file-level protection of data at rest on storage devices. With RDMA protocols and physical memory, no such stacks are present. And, to add such support would introduce CPU processing and its inherent latency, counter to the goals of the remote storage approach. Requiring the peer to verify by remotely reading the data is prohibitive in both bandwidth and latency, and without additional mechanisms to ensure the actual stored data is read (and not a copy in some volatile cache), can not provide the necessary result.

To address this, an integrity operation is required. The integrity check is initiated by the upper layer or application, which optionally computes the expected hash of a given segment of arbitrary size, sending the hash via an RDMA Verify operation targeting the RDMA segment on the responder, and the responder calculating and optionally verifying the hash on the indicated data, bypassing any volatile copies remaining in caches. The responder responds with its computed hash value, or optionally, terminates the connection with an appropriate error status upon mismatch. Specifying this optional

termination behavior enables a transaction to be sent as WRITE-FLUSH-VERIFY-ATOMICWRITE, without any pipeline bubble. The result (carried by the subsequently ordered ATOMIC_WRITE) will not not be committed as valid if any prior operation is terminated, and in this case, recovery can be initiated by the requestor immediately from the point of failure. On the other hand, an errorless "scrub" can be implemented without the optional termination behavior, by providing no value for the expected hash. The responder will return the computed hash of the contents.

The hash algorithm is not specified by the RDMA protocol, instead it is left to the upper layer to select an appropriate choice based upon the strength, security, length, support by the RNIC, and other criteria. The size of the resulting hash is therefore also not specified by the RDMA protocol, but is dictated by the hash algorithm. The RDMA protocol becomes simply a transport for exchanging the values.

It should be noted that the design of the operation, passing of the hash value from requestor to responder (instead of, for example, computing it at the responder and simply returning it), allows both peers to determine immediately whether the segment is considered valid, permitting local processing by both peers if that is not the case. For example, a known-bad segment can be immediately marked as such ("poisoned") by the responder platform, requiring recovery before permitting access. [cf ACPI, JEDEC, SNIA NVMP specifications]

2.4. Local Semantics

The new operations imply new access methods ("verbs") to local persistent memory which backs registrations. Registrations of memory which support persistence will follow all existing practices to ensure permission-based remote access. The RDMA protocols do not expose these permissions on the wire, instead they are contained in local memory registration semantics. Existing attributes are Remote Read and Remote Write, which are granted individually through local registration on the machine. If an RDMA Read or RDMA Write operation arrives which targets a segment without the appropriate attribute, the connection is terminated.

In support of the new operations, new memory attributes are needed. For RDMA Flush, two "Flushable" attributes provide permission to invoke the operation on memory in the region for persistence and/or global visibility. When registering, along with the attribute, additional local information can be provided to the RDMA layer such as the type of memory, the necessary processing to make its contents persistent, etc. If the attribute is requested for memory which cannot be persisted, it also allows the local provider to return an

error to the upper layer, obviating the upper layer from providing the region to the remote peer.

For RDMA Verify, the "Verifiable" attribute provides permission to compute the hash of memory in the region. Again, along with the attribute, additional information such as the hash algorithm for the region is provided to the local operation. If the attribute is requested for non-persistent memory, or if the hash algorithm is not available, the local provider can return an error to the upper layer. In the case of success, the upper layer can exchange the necessary information with the remote peer. Note that the algorithm is not identified by the on-the-wire operation as a result. Establishing the choice of hash for each region is done by the local consumer, and each hash result is merely transported by the RDMA protocol. Memory can be registered under multiple regions, if differing hashes are required, for example unique keys may be provisoned to implement secure hashing. Also note that, for certain "reversible" hash algorithms, this may allow peers to effectively read the memory, therefore, the local platform may require additional read permissions to be associated with the Verifiable permission, when such algorithms are selected.

The Atomic Write operation requires no new attributes, however it does require the "Remote Write" attribute on the target region, as is true for any remotely requested write. If the Atomic Write additionally targets a Flushable region, the RDMA Flush is performed separately. It is never generally possible to achieve persistence atomically with placement, even locally.

3. RDMA Protocol Extensions

The extensions in this document fall into two categories:

- Protocol extensions
- Local behavior extensions

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

3.1. RDMAP Extensions

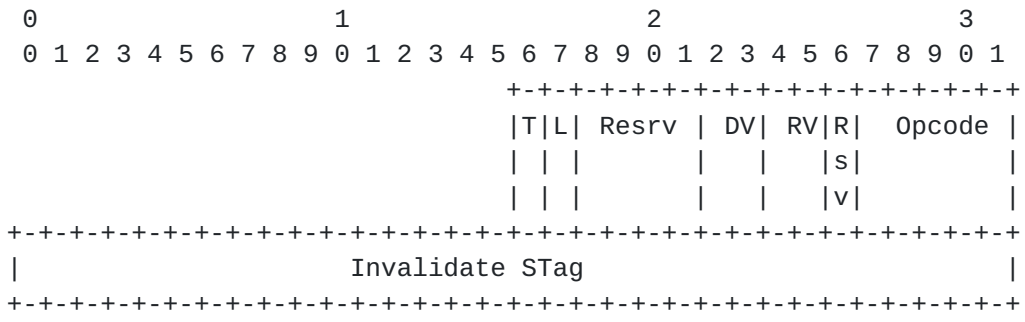
The wire-related aspects of the extensions are discussed in this section.This document defines the following new RDMA operations.

For reference, Figure 1 depicts the format of the DDP Control and RDMAP Control Fields, in the style and convention of RFC5040 and RFC7306:

The DDP Control Field consists of the T (Tagged), L (Last), Resrv, and DV (DDP protocol Version) fields are defined in RFC5041. The RDMAP Control Field consists of the RV (RDMA Version), Rsv, and Opcode fields are defined in RFC5040. No change or extension is made to these fields by this specification.

This specification adds values for the RDMA Opcode field to those specified in RFC5040. 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.



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 2.
- 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 Flush Request	0	N/A	1	opt	Yes
01101b	RDMA Flush Response	0	N/A	3	N/A	No
01110b	RDMA Verify Request	0	N/A	1	opt	Yes
01111b	RDMA Verify Response	0	N/A	3	N/A	Yes
10000b	Atomic Write Request	0	N/A	1	opt	Yes
10001b	Atomic Write Response	0	N/A	3	N/A	No

Additional RDMA Usage of DDP Fields

This extension adds RDMAP use of Queue Number 1 for Untagged Buffers for issuing RDMA Flush, RDMA Verify and Atomic Write Requests, and use of Queue Number 3 for Untagged Buffers for tracking the respective Responses.

All other DDP and RDMAP Control Fields are set as described in RFC5040 and RFC7306.

Table 3 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 Flush Request	None	No
01101b	RDMA Flush Response	None	No
01110b	RDMA Verify Request	None	No
01111b	RDMA Verify Response	None	No
10000b	Atomic Write Request	None	No
10000b	Atomic Write Response	None	No

RDMA Message Definitions

3.1.1. RDMA Flush

The RDMA Flush operation requests that all bytes in a specified region are to be made persistent and/or globally visible, under control of specified flags. As specified in section 4 its operation is ordered after the successful completion of any previous requested RDMA Write or certain other operations. The response is generated after the region has reached its specified state. The implementation MUST fail the operation and send a terminate message if the RDMA Flush cannot be performed, or has encountered an error.

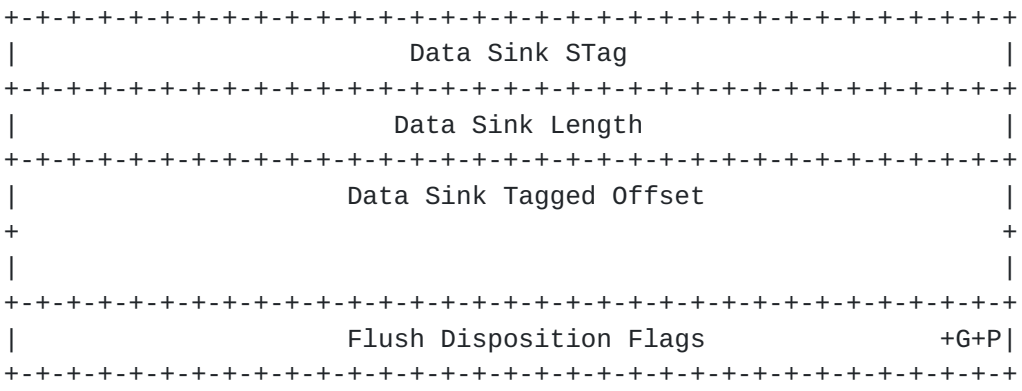
The RDMA Flush operation MUST NOT be completed by the data sink until all data has attained the requested state. Achieving persistence may require programming and/or flushing of device buffers, while achieving global visibility may require flushing of cached buffers across the entire platform interconnect. In no event are persistence and global visibility achieved atomically, one may precede the other and either may complete at any time.The Atomic Write operation may be used by an upper layer consumer to indicate that either or both

dispositions are available after completion of the RDMA Flush, in addition to other approaches.

3.1.1.1. RDMA Flush Request Format

The RDMA Flush Request Message makes use of the DDP Untagged Buffer Model. RDMA Flush 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 RDMA Flush Requests allows the operations to reuse the same RDMA infrastructure (e.g. Outbound and Inbound RDMA Read Queue Depth (ORD/IRD) flow control) as that defined for RDMA Read Requests.

The RDMA Flush Request Message carries a payload that describes the ULP Buffer address in the Responder's memory. The following figure depicts the Flush Request that is used for all RDMA Flush Request Messages:



Flush Request

Data Sink STag: 32 bits The Data Sink STag identifies the Remote Peer's Tagged Buffer targeted by the RDMA Flush 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 Length: The Data Sink Length is the length, in octets, of the bytes targeted by the RDMA Flush Request.

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 Flush Request.

Flags: Flags specifying the disposition of the flushed data: 0x01 Flush to Persistence, 0x02 Flush to Global Visibility.

3.1.1.2. RDMA Flush Response

The RDMA Flush Response Message makes use of the DDP Untagged Buffer Model. RDMA Flush Response messages MUST use the same Queue Number as RDMA Extensions Atomic Operation Responses (QN=3). No payload is passed to the DDP layer on Queue Number 3.

Upon successful completion of RDMA Flush processing, an RDMA Flush Response MUST be generated.

If during RDMA Flush processing on the Responder, an error is detected which would result in the requested region to not achieve the requested disposition, the Responder MUST generate a Terminate message. The contents of the Terminate message are defined in Section 5.2.

3.1.1.3. RDMA Flush Ordering and Atomicity

Ordering and completion rules for RDMA Flush Request are similar to those for an Atomic operation as described in section 5 of RFC7306. The queue number field of the RDMA Flush Request for the DDP layer MUST be 1, and the RDMA Flush Response for the DDP layer MUST be 3.

There are no ordering requirements for the placement of the data, nor are there any requirements for the order in which the data is made globally visible and/or persistent. Data received by prior operations (e.g. RDMA Write) MAY be submitted for placement at any time, and persistence or global visibility MAY occur before the flush is requested. After placement, data MAY become persistent or globally visible at any time, in the course of operation of the persistency management of the storage device, or by other actions resulting in persistence or global visibility.

Any region segment specified by the RDMA Flush operation MUST be made persistent and/or globally visible before successful return of the operation. If RDMA Flush processing is successful on the Responder, meaning the requested bytes of the region are, or have been made persistent and/or globally visible, as requested, the RDMA Flush Response MUST be generated.

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

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

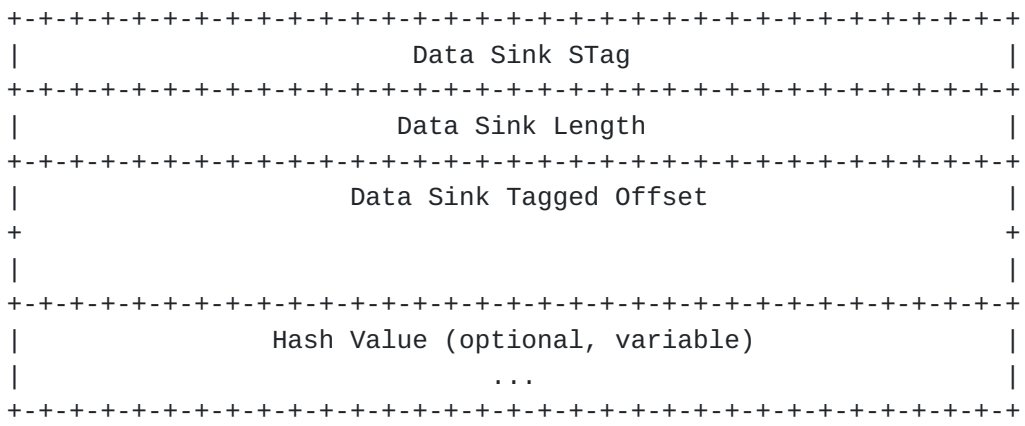
3.1.2. RDMA Verify

The RDMA Verify operation requests that all bytes in a specified region are to be read from the underlying storage and that an integrity hash be calculated. As specified in section 4 its operation is ordered after the successful completion of any previous requested RDMA Write and RDMA Flush, or certain other operations. The implementation MUST fail the operation and send a terminate message if the RDMA Verify cannot be performed, has encountered an error, or if a hash value was provided in the request and the calculated hash does not match. If no condition for a Terminate message is encountered, the response is generated containing the result calculated hash value.

3.1.2.1. RDMA Verify Request Format

The RDMA Verify Request Message makes use of the DDP Untagged Buffer Model. RDMA Verify 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 RDMA Read and RDMA Flush Requests allows the operations to reuse the same RDMA infrastructure (e.g. Outbound and Inbound RDMA Read Queue Depth (ORD/IRD) flow control) as that defined for those requests.

The RDMA Verify Request Message carries a payload that describes the ULP Buffer address in the Responder's memory. The following figure depicts the Verify Request that is used for all RDMA Verify Request Messages:



Verify Request

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

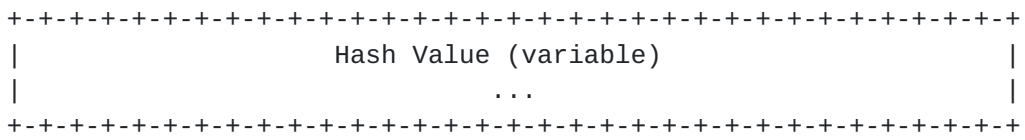
Data Sink Length: The Data Sink Length is the length, in octets, of the bytes targeted by the Verify Request.

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 Verify Request.

Hash Value: The Hash Value is optionally an octet string representing the expected result, if any, of the hash algorithm on the Remote Peer's Tagged Buffer. The length of the Hash Value is variable, and dependent on the selected algorithm. When provided, any mismatch with the calculated value causes the Responder to generate a Terminate message, and close the connection. The contents of the Terminate message are defined in section 5.2.

3.1.2.2. Verify Response Format

The Verify Response Message makes use of the DDP Untagged Buffer Model. Verify Response messages MUST use the same Queue Number as RDMA Flush Responses (QN=3). The RDMAP layer passes the following payload to the DDP layer on Queue Number 3. The RDMA Verify Response is not sent when a Terminate message is generated through specifying the Compare Flag as 1, and a mismatch occurs.



Verify Response

Hash Value: The Hash Value is an octet string representing the result of the hash algorithm on the Remote Peer's Tagged Buffer. The length of the Hash Value is variable, and dependent on the algorithm selected by the upper layer consumer, among those supported by the RNIC.

3.1.2.3. RDMA Verify Ordering

Ordering and completion rules for RDMA Verify Request are similar to those for an Atomic operation as described in section 5 of RFC7306. The queue number field of the RDMA Verify Request for the DDP layer MUST be 1, and the RDMA Verify Response for the DDP layer MUST be 3.

As specified in section 4, RDMA Verify and RDMA Flush are executed by the Data Sink in strict order. When an RDMA Verify follows an RDMA Flush, and because the RDMA Flush MUST ensure that all bytes are in the specified state before responding, any RDMA Verify that follows can be assured that it is operating on flushed data. If unflushed data has been sent to the region segment between the operations, and since data may be made persistent and/or globally visible by the Data Sink at any time, the result of any such RDMA Verify is undefined.

3.1.3. Atomic Write

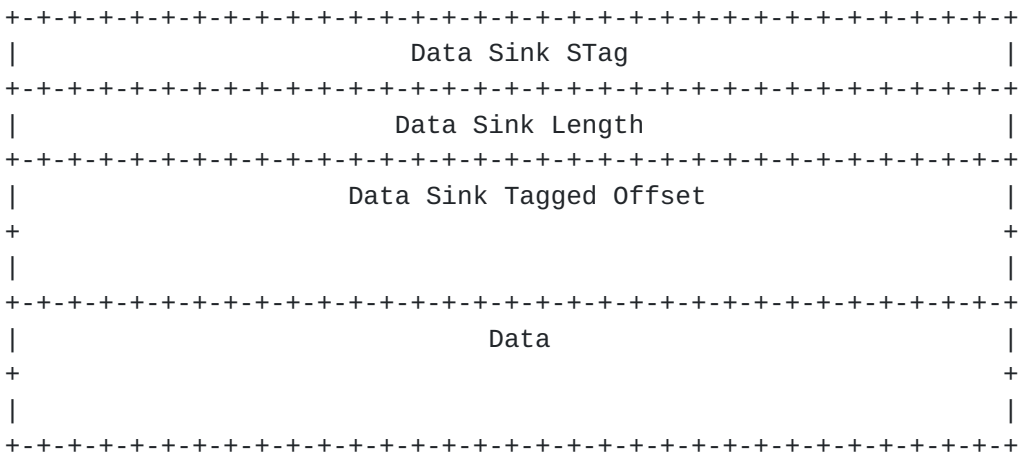
The Atomic Write operation provides a block of data which is placed to a specified region atomically, and as specified in section 4 its placement is ordered after the successful completion of any previous requested RDMA Flush or RDMA Verify. This specified region is constrained in size and alignment to 64-bits at 64-bit alignment, and the implementation MUST fail the operation and send a terminate message if the placement cannot be performed atomically.

The Atomic Write Operation requires the Responder to write a 64-bit value at a ULP Buffer address that is 64-bit aligned in the Responder's memory, in a manner which is Placed in the responder's memory atomically.

3.1.3.1. Atomic Write Request

The Atomic Write Request Message makes use of the DDP Untagged Buffer Model. Atomic Write 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 RDMA Flush and RDMA Verify Requests allows the operations to reuse the same RDMA infrastructure (e.g. Outbound and Inbound RDMA Read Queue Depth (ORD/IRD) flow control) as that defined for those Requests.

The Atomic Write Request Message carries an Atomic Write Request payload that describes the ULP Buffer address in the Responder's memory, as well as the data to be written. The following figure depicts the Atomic Write Request that is used for all Atomic Write Request Messages:



Atomic Write Request

Data Sink STag: 32 bits The Data Sink STag identifies the Remote Peer's Tagged Buffer targeted by the Atomic Write 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 Length: The Data Sink Length is the length of data to be placed, and MUST be 8.

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 Atomic Write Request. This offset can be any value, but the destination ULP buffer address MUST be aligned as specified above. Ensuring that the STag and Data Sink Tagged Offset values appropriately meet such a requirement is an upper layer consumer responsibility, and is out of scope for this specification.

Data: The 64-bit data value to be written, in big-endian format.

Atomic Write Operations MUST target ULP Buffer addresses that are 64-bit aligned, and conform to any other platform restrictions on the Responder system. The write MUST NOT be Placed prior to all prior RDMA Flush operations, and therefore all other prior operations, completing successfully.

If an Atomic Write Operation is attempted on a target ULP Buffer address that is not 64-bit aligned, or due to alignment, size, or other platform restrictions cannot be performed atomically:

The operation MUST NOT be performed

The Responder's memory MUST NOT be modified

A terminate message MUST be generated. (See Section 5.2 for the contents of the terminate message.)

3.1.3.2. Atomic Write Response

The Atomic Write Response Message makes use of the DDP Untagged Buffer Model. Atomic Write Response Response messages MUST use the same Queue Number as RDMA Flush Responses (QN=3). The RDMAP layer passes no payload to the DDP layer on Queue Number 3.

3.1.4. Discovery of RDMAP Extensions

As for RFC7306, 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 Flush and Atomic Write, 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. For RDMA Verify, such an operation may target a buffer with remote read permission.

3.2. Local Extensions

This section discusses memory registration, new memory and protection attributes, and applicability to both remote and "local" (receives). Because this section does not specify any wire-visible semantic, it is entirely informative.

3.2.1. 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 - robust design ensuring future interop

New local PMEM memory registration example:

```
Register(region[], MemPerm, MemType, MemMode) -> STag
```

Region describes the memory segment[s] to be registered by the returned STag. The local RNIC may limit the size and number of these segments.

MemPerm to indicate permitted operations in addition to remote read and remote werite: "remote flush to persistence", "remote flush to global visibility", selectivity, etc.

MemType includes type of storage described by the Region, i.e. plain RAM, "flush required" (flushable), or PCIe-resident via peer-to-peer, or any other local platform-specific processing

MemMode includes disposition of data Read and/or written e.g. Cacheable after operation (indicate if needed by CPU on data sink, to allow or avoid writethrough as optimization)

None of the above attributes are at all relevant, or exposed, by the protocol

STag is processed in receiving RNIC during RDMA operation to specified region, under control of original Perm, Type and Mode.

3.2.2. Completion Semantics

Discuss the interactions with new operations when upper layer provides Completions to responder (e.g. messages via receive or immediate data via RDMA Write). Natural conclusion of ordering rules, but made explicit.

Ordering of operations is critical: Such RDMA Writes cannot be allowed to "pass" persistence or global visibility, and RDMA Flush may not begin until prior RDMA Writes to flush region are accounted for. Therefore, ULP protocol implications may also exist.

3.2.3. Platform Semantics

Writethrough behavior on persistent regions and reasons for same. Consider recommending a local writethrough behavior on any persistent region, to support a nonblocking hurry-up to avoid future stalls on a subsequent cache flush, prior to a flush. Also, it would enhance storage integrity. Drive selection of this behavior from memory registration, so RNIC may "look up" the desired behavior in its TPT.

PCI extension to support Flush would allow RNIC to provide persistence and/or global visibility 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) or some other flow tag. Or if RNIC participates in platform consistency domain on memory bus or within CPU complex... other possibilities exist!

Also consider additional "integrity check" behavior (hash algorithm) specified per-region. If so, providing it as a registration parameter enables fine-graned control, and enables storing it in per-region RNIC state, making its processing optional and straightforward.

A similar approach applicable to providing security key for encrypting/decrypting access on per-region basius, without protocol exposure. [SDC2017 presentation]

Any other per-region processing to be explored.

4. Ordering and Completions Table

The table in this section specifies the ordering relationships for the operations in this specification and in those it extends, 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.

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 Flush	TODO	No Placement Guarantee between Foo and Bar	N/A	Completed in Order
TODO	RDMA Flush	Placement Guarantee between Foo and Bar	N/A	TODO
TODO	TODO	Etc	Etc	Etc

Ordering of Operations

5. Error Processing

In addition to error processing described in section 7 of RFC5040 and section 8 of RFC7306, 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 an RDMA Flush, RDMA Verify or Atomic Write 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 Flush, RDMA Verify or Atomic Write Request or 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 Flush or RDMA Verify itself, that is, the act of flushing or verifying the data, the Terminate Message is generated as per the referenced specifications. Even though data is not lost, the upper layer MUST be notified of the failure by informing the requester of the status, terminating any queued operations, and allow the requester to perform further action, for instance, recovery.

5.2. Errors Detected at the Remote Peer

On incoming RDMA Flush and RDMA Verify 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 Flush, RDMA Verify or Atomic Write operation cannot be satisfied, due to transient or permanent errors detected in the processing by the Responder, a Terminate message MUST be returned to the Requestor.

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 Flush Request, this specification

0xD RDMA Flush Response, this specification

0xE RDMA Verify Request, this specification

0xF RDMA Verify Response, this specification

0x10 Atomic Write Request, this specification

0x11 Atomic Write Response, 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 RFC5040 and RDMA Protocol Extensions in RFC7306, and as such the Security Considerations discussed in Section 8 of RFC5040 and

Section 9 of RFC7306 apply. In particular, all operations use ULP Buffer addresses for the Remote Peer Buffer addressing used in RFC5040 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 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 persistence and/or global visibility, the lack of such an approach would lead to frequent recovery in low-resource situations, potentially opening a new threat to such applications.

8. To Be Added or Considered

This section will be deleted in a future document revision.

Complete the discussion in section 3.2 and its subsections, Local Extension semantics.

Complete the Ordering table in section 4. Carefully include discussion of the order of "start of execution" as well as completion, which are somewhat more involved than prior RDMA operation ordering.

RDMA Flush "selectivity", to provide default flush semantics with broader scope than region-based. If specified, a flag to request that all prior write operations on the issuing Queue Pair be flushed with the requested disposition(s). This flag may simplify upper layer processing, and would allow regions larger than 4GB-1 byte to be flushed in a single operation. The STag, Offset and Length will be ignored in this case. It is to-be-determined how to extend the RDMA security model to protect other regions associated with this Queue Pair from unintentional or unauthorized flush.

9. Acknowledgements

The authors wish to thank Jim Pinkerton, who contributed to an earlier version of the specification, and Brian Hausauer and Kobby Carmona, who have provided significant review and valuable comments.

10. References

10.1. 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, <https://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, <https://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, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc7306>.

10.2. 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, <https://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, <https://www.rfc-editor.org/info/rfc5661>.

[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, <https://www.rfc-editor.org/info/rfc7145>.

[RFC8166] Lever, C., Ed., Simpson, W., and T. Talpey, "Remote Direct Memory Access Transport for Remote Procedure Call Version 1", RFC 8166, DOI 10.17487/RFC8166, June 2017, <https://www.rfc-editor.org/info/rfc8166>.

[RFC8267] Lever, C., "Network File System (NFS) Upper-Layer Binding to RPC-over-RDMA Version 1", RFC 8267, DOI 10.17487/RFC8267, October 2017, <https://www.rfc-editor.org/info/rfc8267>.

[SCSI] ANSI, "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)", March 2020.

https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-smb2/5606ad47-5ee0-437a-817e-70c366052962

[SMBDirect] Microsoft Corporation, "SMB2 Remote Direct Memory Access (RDMA) Transport Protocol (MS-SMBD)", September 2018.

https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-smbd/1ca5f4ae-e5b1-493d-b87d-f4464325e6e3

[SNIANVMP] SNIA NVM Programming TWG, "SNIA NVM Programming Model v1.2", June 2017.

https://www.snia.org/sites/default/files/technical_work/final/NVMProgrammingModel_v1.2.pdf

10.3. URIs

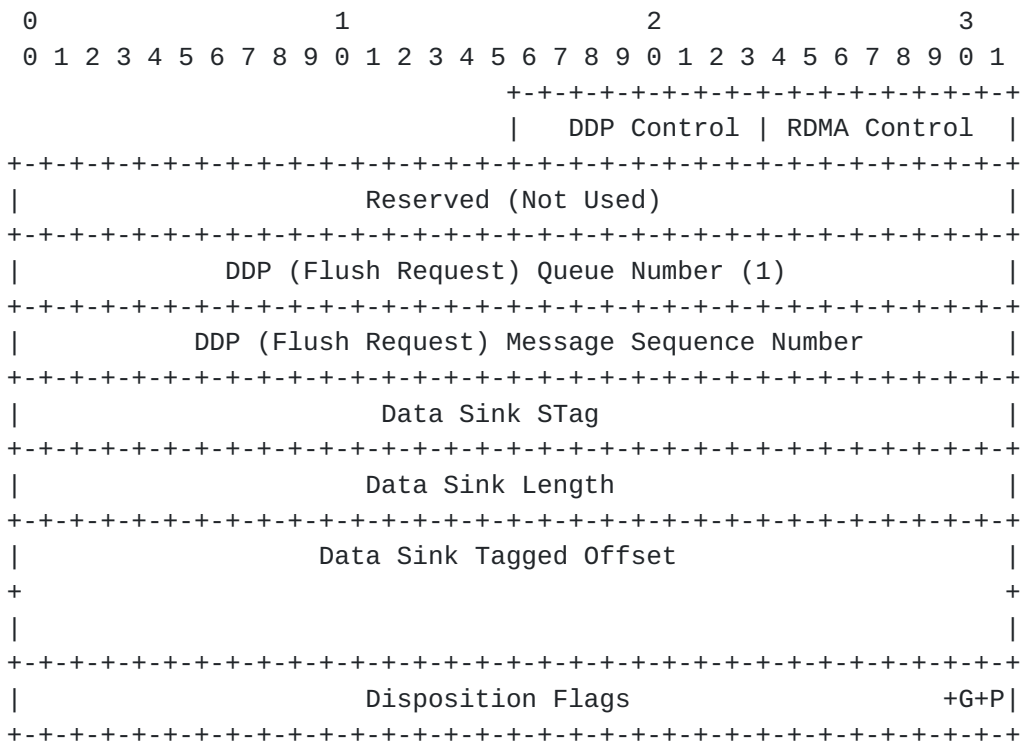
[1] http://www.nvmexpress.org

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

Appendix A. DDP Segment Formats for RDMA Extensions

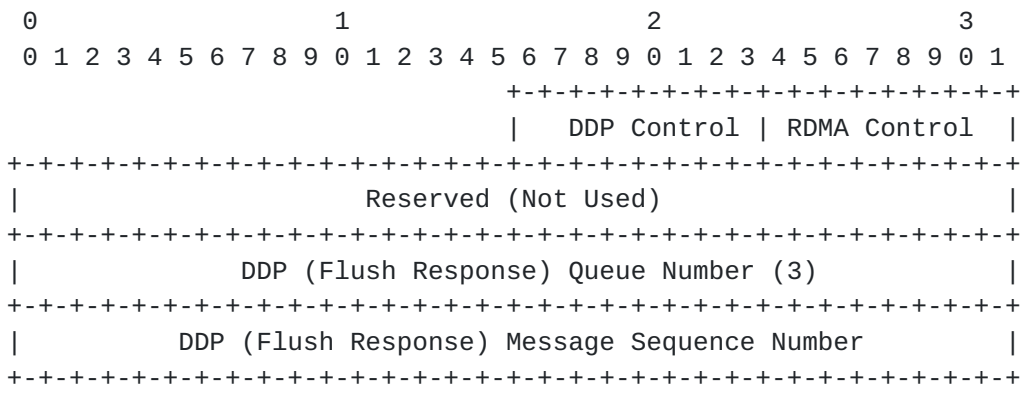
This appendix is for information only and is NOT part of the standard. It simply depicts the DDP Segment format for each of the RDMA Messages defined in this specification.

A.1. DDP Segment for RDMA Flush Request



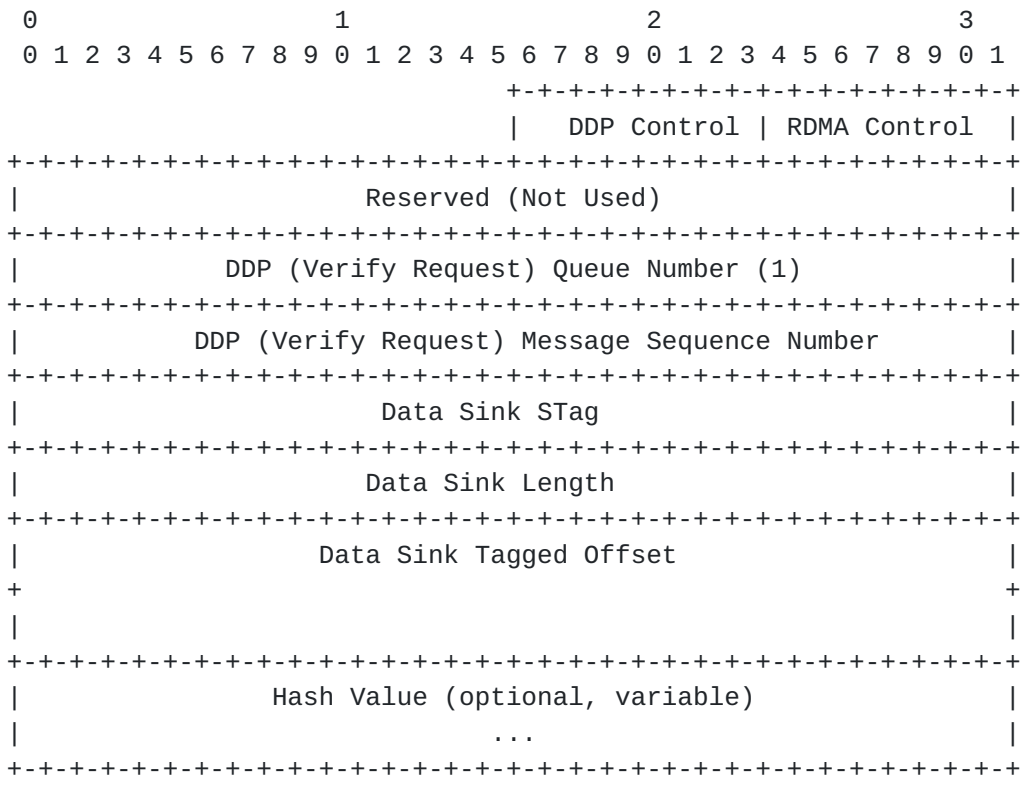
RDMA Flush Request, DDP Segment

A.2. DDP Segment for RDMA Flush Response



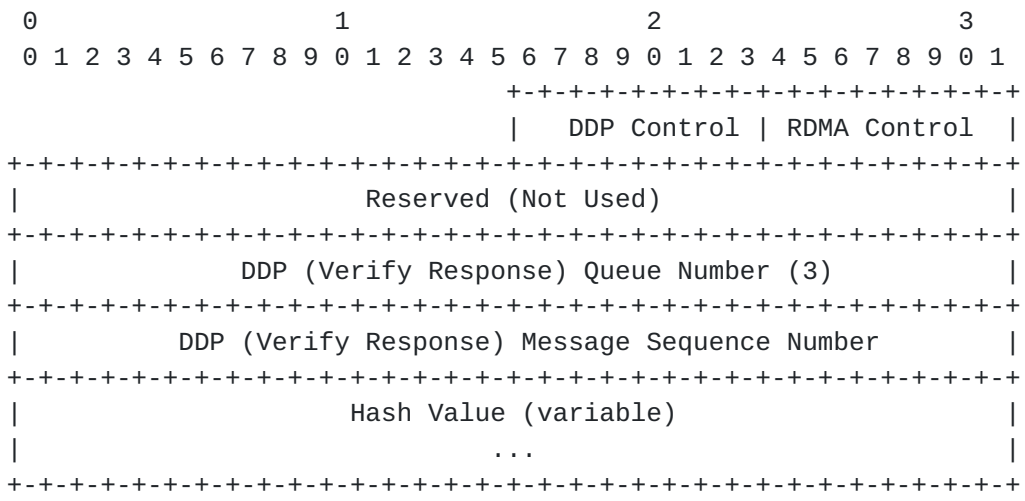
RDMA Flush Response, DDP Segment

A.3. DDP Segment for RDMA Verify Request



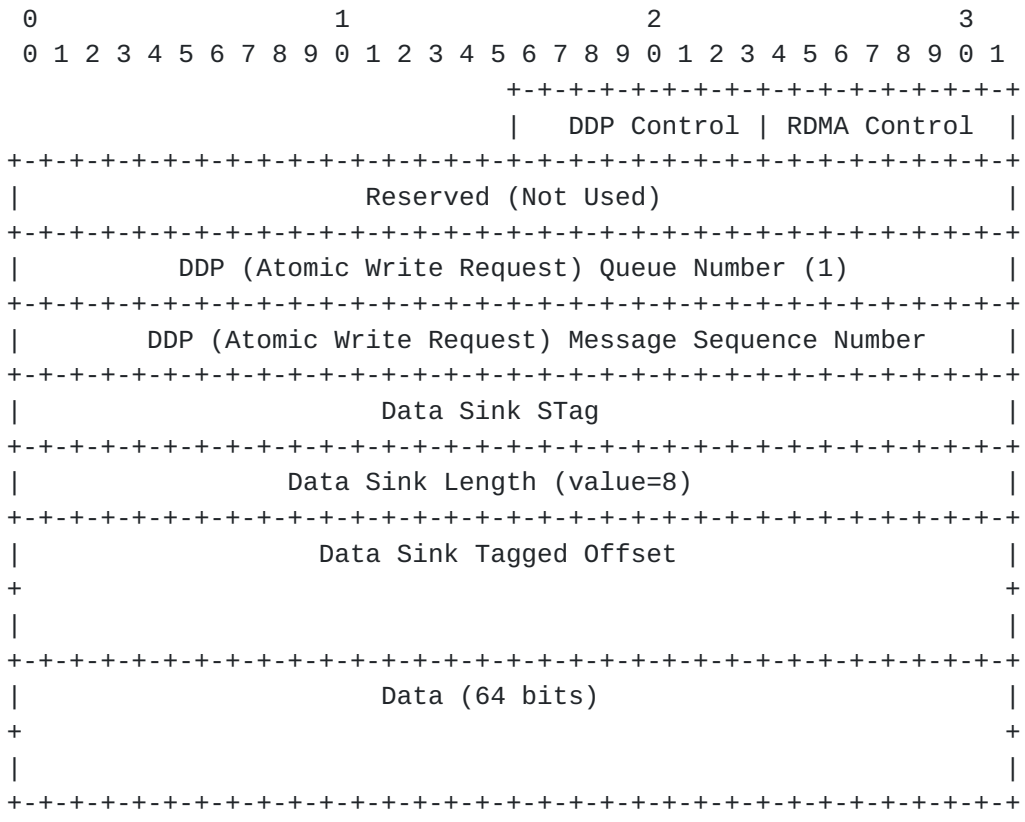
RDMA Verify Request, DDP Segment

A.4. DDP Segment for RDMA Verify Response



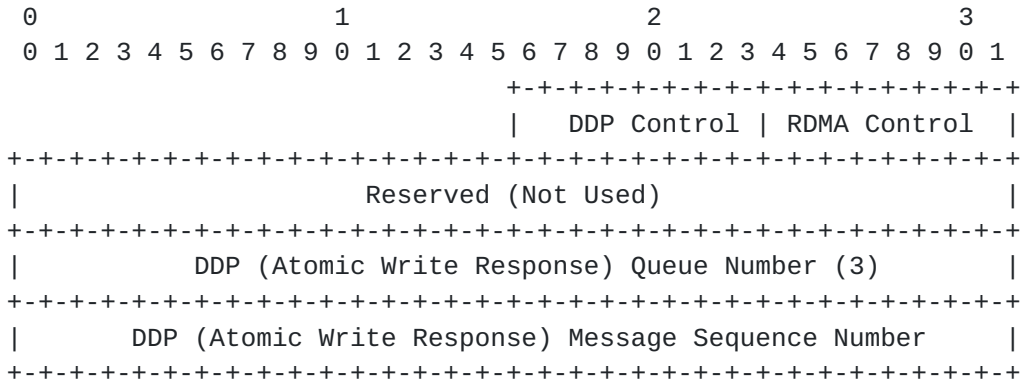
RDMA Verify Response, DDP Segment

A.5. DDP Segment for Atomic Write Request



Atomic Write Request, DDP Segment

A.6. DDP Segment for Atomic Write Response



Atomic Write Response, DDP Segment

Authors' Addresses

Tom Talpey
Microsoft
One Microsoft Way
Redmond, WA 98052
US

Email: ttalpey@microsoft.com

Tony Hurson
Intel
Austin, TX
US

Email: tony.hurson@intel.com

Gaurav Agarwal
Marvell
CA
US

Email: gagarwal@marvell.com

Tom Reu
Chelsio
NJ
US

Email: tomreu@chelsio.com