

***Proposed*** OFA libfabric API extensions to add native RDMA support for persistent memory

Primary Author: Chet Douglas (chet.r.douglas@intel.com)

Organization: DCG Crystal Ridge SW Architecture

Date: 06-07-2016

Version: V0.62

Table of Contents

[1 Document Revision History 2](#_Toc453054634)

[2 Document Overview 3](#_Toc453054635)

[1. libfabric API 3](#_Toc453054636)

[3 Motivations for this proposal 4](#_Toc453054637)

[4 OFA High-Level Software Interfaces 5](#_Toc453054638)

[1. Proposed libfabric Interface extensions for PMEM 5](#_Toc453054639)

[4.1.1 fi\_getinfo 5](#_Toc453054640)

[4.1.2 fi\_mr\_reg 6](#_Toc453054641)

[4.1.3 fi\_writemsg Updates 7](#_Toc453054642)

[4.1.4 fi\_writemsg ordering and completion semantics with PMEM 8](#_Toc453054643)

[4.1.5 Sample Use Cases 9](#_Toc453054644)

[5 Software API & Architecture Opens 14](#_Toc453054645)

Table of Figures

[Figure 1- Current libfabric API usage with PMEM 9](#_Toc453054765)

[Figure 2- Proposed libfabric extension usage with PMEM 10](#_Toc453054766)

[Figure 3- Proposed libfabric extension usage with PMEM 12](#_Toc453054767)

[Figure 4- Proposed libfabric extension usage with PMEM w "non-standard" memory device 13](#_Toc453054768)

# Document Revision History

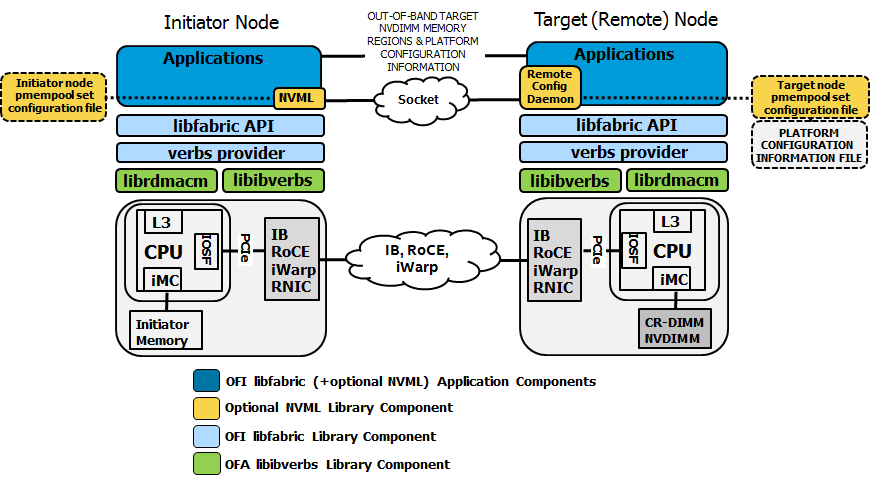
|  |  |
| --- | --- |
| **Version** | **Document Changes** |
| V0.60 05/11/16 | -Initial document with updates for internal Intel libfabric interface reviews -Added open issues section to track additional work for this proposal |
| V0.61 05/19/16 | -Updates from the SNIA NVM TWG review  -Fixed use case pictures and added missing cases  -Updates to open architecture section with NVM TWG review feedback added |
| V0.62 06/07/16 | -Added open architecture section consideration to create new fi\_\* API instead of overloading fi\_writemsg with FI\_COMMIT and FI\_IMMED  -Moved opens section to the end of the doc -Updates from latest SNIA review & discussion -Updates to open section from first OFA DSDA open discussion |

# Document Overview

This document describes proposed Open Fabric Alliance (OFA) SW API extensions/modifications to support native accesses utilizing remote byte addressable persistent memory. The scope is limited to the OFA libfabric and libibverb network access application libraries. This document only describes the changes related to adding remote persistent memory support and does not document all of the current interfaces that are not related to these changes.

## libfabric API

libfabric API is a Linux ring3 application common network fabric API that is being introduced in to the Linux community and is governed by the OFA OFIWG. A kernel version of the API is also being worked on and is governed by OFA DSDA WG. Behind the API is a set of libfabric providers that implement the API for each fabric technology. In the picture below a verbs provider implements the libfabric API and provides a thin layer to the existing libibverbs library. This provides libfabric API linkage to RoCE, IB, and iWARP based fabrics.



# Motivations for this proposal

* This proposal is primarily driven by a detailed Intel HW assessment of the RDMA IO paths and an understanding of how the Intel chipset architecture currently works
* The motivation for these SW API extensions is to increase bandwidth and obtain the lowest possible latency with the lowest HW design complexity and reflects Intel’s current thinking on how we might implement native pmem support for RDMA
* We have talked to a significant number of ISVs and OSVs to try and understand the most common Use Cases from an application perspective and have incorporated the feedback in to this proposal
* There are a number of open architecture questions in the proposed API extensions where we need more detailed feedback and more detailed understanding of application Use Cases
* The main goal of publicizing this SW API proposal now is to start an open discussion & dialog across the industry with the goal to eventually end with a set of standardized OFA API Extensions to provide native RDMA support with pmem

# OFA High-Level Software Interfaces

## Proposed libfabric Interface extensions for PMEM

The following sections describe the proposed modifications and extension to the Linux libfabric Interface Extensions.

### fi\_getinfo

-Retrieve information about each fabric endpoint for a given connection  
-Capabilities expanded to describe byte addressable persistent memory support for each endpoint

**int fi\_getinfo( int version,   
 const char \*node,   
 const char \*service,   
 uint64\_t flags,   
 struct fi\_info \*hints,   
 struct fi\_info \*\*info);**   
  
Add the following capability bit to the ***info flags*** field to describe additional capabilities of the system. The caller requests specific capabilities to be supported in the ***info*** struct and the endpoint SW will update the same ***info*** struct with the supported capabilities:

* **FI\_PMEM**
  + Initiator or target node is capable of supporting byte addressable persistent memory (Pmem). Assumes endpoint supports read/write to pmem with additional write flags utilized in fi\_writemsg and additional access flags utilized in fi\_mr\_reg.
  + Endpoints that only support a single read or write direction to pmem can optional set the FI\_READ, FI\_READ\_REMOTE, or FI\_WRITE, FI\_WRITE\_REMOTE flags with FI\_PMEM to report specific direction supported.

Alter the following to the current flags definitions:

* **FI\_RMA\_EVENT :** Requests that an endpoint support the generation of completion events when it is the target of an RMA and/or atomic operation. If set, the provider will support both completion queue and counter events. This flag requires that FI\_REMOTE\_READ and/or FI\_REMOTE\_WRITE **and/or FI\_PMEM** be enabled on the endpoint.

### fi\_mr\_reg

-Register persistent memory data buffer addresses with the fabric controller for a specific protection domain with requested accesses and return the lkey and rkey handles that describe the registered memory

OPEN: Tweak the language to reflect the current use of keys..  
-Access attributes expanded to include new memory and device types

-By making these part of opaque rkey, initiator SW is not burdened with understanding these attributes

**int fi\_mr\_reg( struct fid\_domain \* domain,**

**const void \* buf,**

**size\_t len,**

**uint64\_t access,**

**uint64\_t offset,**

**uint64\_t requested\_key,**

**uint64\_t flags,**

**struct fid\_mr \*\* mr,**

**void \* context);**

Add the following bits to the ***flags*** field to further describe the attributes of the memory region being registered. The access requested is a combination of OR’ing these new access capabilities with existing flags:

* **FI\_PMEM** – Memory region being registered is byte addressable persistent memory
* **FI\_UNCACHED** - Memory region should not be backed by cache. When data is written to this region, the local cpu caches should be bypassed. Without this flag being present, the write data should be placed in cpu cache as SW will most likely access the data shortly after remote transfer is complete.
* **FI\_NON\_STANDARD\_MEMORY\_DEVICE** - Memory region is resident on a device attached to a bus not typically utilized for memory, like a PCI device or PCI NTB. These devices typically require additional memory resources like MMIO BAR and mailbox addresses that will be utilized by the target endpoint to complete the write transaction. As a result of SW handling this API, kernel components would be required to discover the additional addresses and virtual mappings and supply them to the RNIC as part of the memory range registration process. The resulting opaque values for the LKEY and RKEY should contain additional context to describe the additional resources but the mechanism to do this is not considered to be part of this high-level network API.

### fi\_writemsg Updates

-The existing fi\_writemsg API is utilized for writing to persistent memory  
-When this command is utilized with the FI\_COMMIT flag, this command has the completion semantics of an fi\_read and will not return a completion to the initiator until all data within scope of the command has been committed to the durable memory domain.   
-The write with commit allows previous write data to be attached to the command. Previous calls to fi\_write\* utilizing the same QP and same RKEY value will be within scope of this write with commit and will also be committed to the durable memory domain before the completion is signaled.  
-The existing libfabric mechanism for setting up a CQ is utilized to set up and register an initiator SW completion queue and notification for writes with commit.

**static inline ssize\_t fi\_writemsg(struct fid\_ep \*ep,**

**const struct fi\_msg\_rma \*msg,**

**uint64\_t flags)**

**struct fi\_msg\_rma {**

**const struct iovec \*msg\_iov;**

**void \*\*desc;**

**size\_t iov\_count;**

**fi\_addr\_t addr;**

**const struct fi\_rma\_iov \*rma\_iov;**

**size\_t rma\_iov\_count;**

**void \*context;**

**uint64\_t data;**

**};**

The following libfabric *flags* are added for handling writes to persistent memory. The flags are utilized by the target node endpoint device to precisely control steering of the write data and handling of any device specific completion handling. Therefore these indicators should be visible in the wire protocol payload and available at the target end point:**FI\_COMMIT** *–* Commit to pmem all data within scope of the command. Completion to the initiator occurs after all data has been committed to the durable memory domain. Previous fi\_write\* messages sent to the same rkey on the same QP will also be committed to durability before the completion is signaled.  
 -With a non-volatile memory region (memory registered with FI\_PMEM), completion indicates all write data in scope has reached durability and is power fail safe. Once durability occurs the Initiator RNIC will insert a Completion WQE on the initiators CQ to notify SW.  
 -With a memory region that is volatile memory (memory registered without FI\_PMEM). The completion indicates all write data in scope has reached the global visibility point   
**FI\_IMMED** – Used in conjunction with FI\_COMMIT – Once all write data in scope of the write has reached the pmem durability domain issue a Completion WQE to the target CQ. Setting this flag without setting FI\_COMMIT is considered an error.  
**FI\_FENCE** – Extend the use of this existing flag to cover fencing of writes on the target node. When set with FI\_COMMIT, the target endpoint will guarantee that previous writes with the same RKEY will be made durable *before* executing this fenced write to the same RKEY.

### fi\_writemsg ordering and completion semantics with PMEM

Here are the basic rules for using the fi\_write\* API with persistent memory:

* All existing libfabric fi\_write\* API can be utilized to write data to PMEM by supplying an RKEY whose memory region was registered with FI\_PMEM set
* fi\_write, fi\_writev, fi\_writedata, fi\_inject\_write and fi\_inject\_writedata API that do not take a *flag* argument cannot request data to be committed to the persistent memory durable domain
* For fi\_writemsg (with FI\_COMMIT set) to properly commit other write data previously sent via fi\_write\* API methods, all of the commands must utilize the same QP for all command submittals and must utilize the same RKEY.
* There is no write data ordering guarantee for any sequence of fi\_write\* or fi\_writemsg commands sent to different QPs or different RKEYs.
* The ordering of write data associated with fi\_writemsg (with FI\_COMMIT set) with respect to the ordering of write data for other fi\_writemsg (with FI\_COMMIT set) requests is indeterminate, even when issued on the same QP and RKEY. It is possible for one to pass the other. SW must utilize the FI\_FENCE with FI\_COMMIT to avoid this indeterminate ordering on the same RKEY. To control write data placement ordering on the same QP but to different RKEYs, SW can continue to utilize fi\_read\* or fi\_send\* API in between fi\_writemsg (with FI\_COMMIT set) commands.

### Sample Use Cases

#### Existing libfabric API being used with PMEM

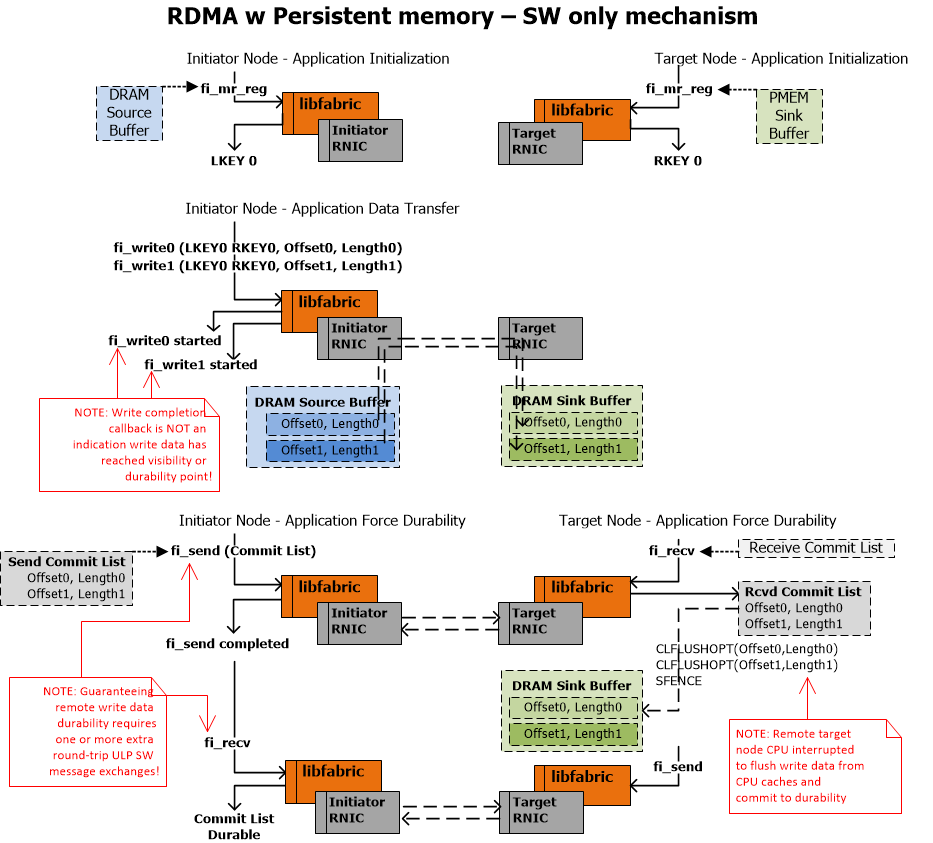


Figure 1- Current libfabric API usage with PMEM

#### Proposed libfabic API extensions being used with PMEM

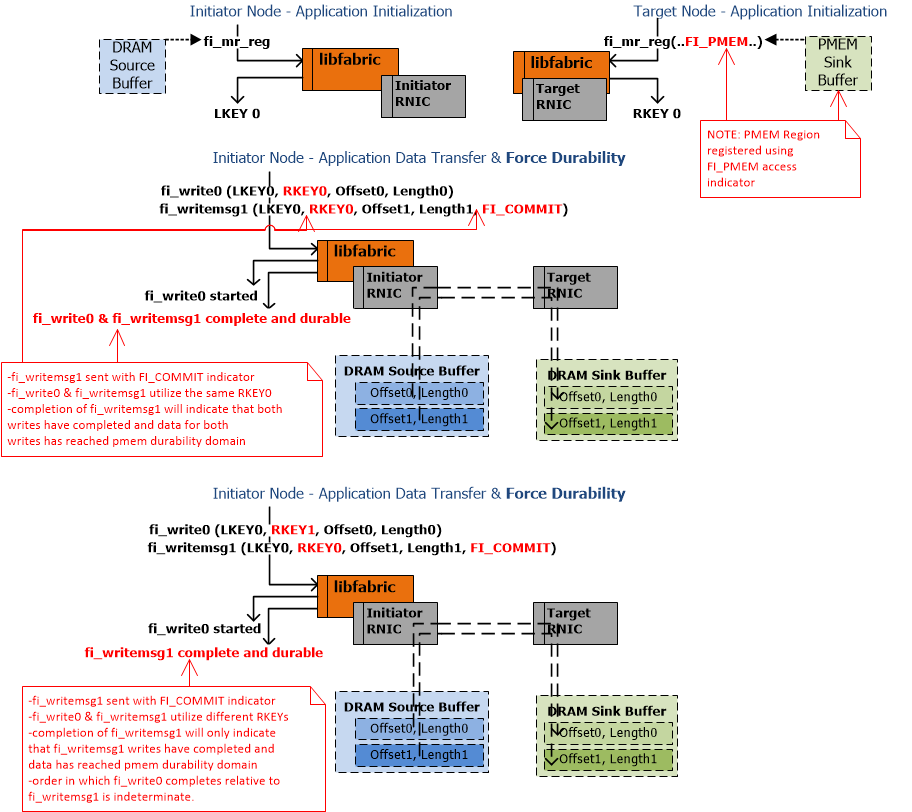
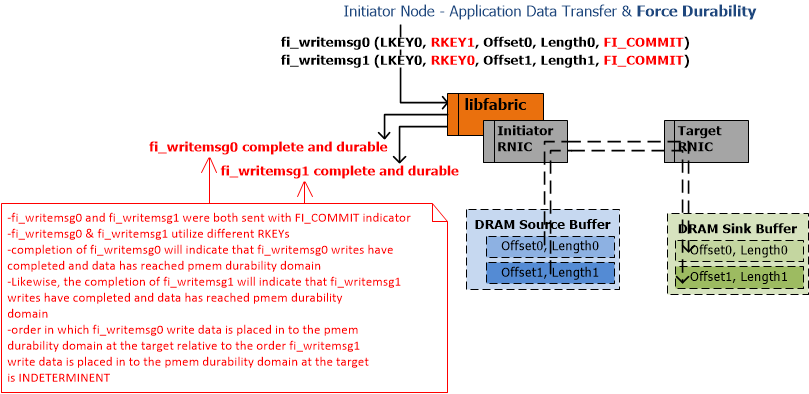


Figure 2- Proposed libfabric extension usage with PMEM



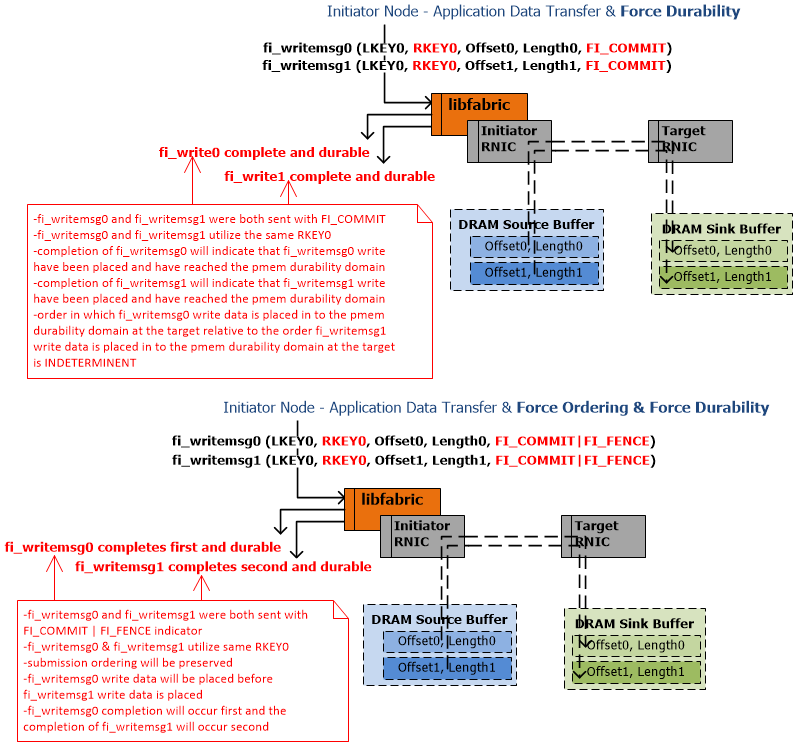


Figure 3- Proposed libfabric extension usage with PMEM

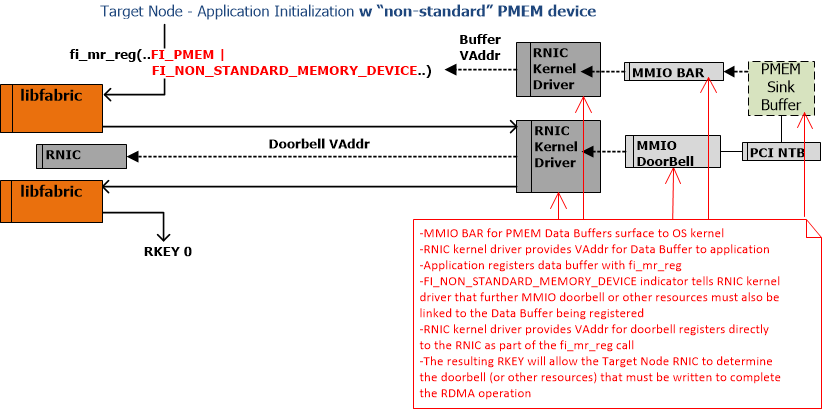


Figure 4- Proposed libfabric extension usage with PMEM w "non-standard" memory device

# Software API & Architecture Opens

This section outlines the open architecture issues effecting the proposed OFA libfabric API:

* **Fencing of Write Commit**
  + This proposed implementation allows optional strict target node write data placement ordering to be imposed at the initiator for the *current* write relative to *previous* writes as long as the writes are all issued on the same QP and RKEY. However, *subsequent* writes can pass previous writes and current write
  + Do we need to consider controlling ordering of the current write with respect to previous writes *AND* *subsequent* future writes? This is not currently in our proposal as it forces in-order data placement which is a complexity that we would like to avoid.
    - **SNIA NVM TWG Feedback:** The current SNIA Programming model implies that applications won’t issue any more writes to a QP/RKEY until the outstanding commit has completed. This means that there won’t be subsequent future writes outstanding when a commit is sent. If SNIA decides to take add multiple sessions/threads to the programming model, then these details would need to be considered.
  + Should we utilize an indicator to allow SW to dynamically apply commit scope to QP with or without RKEY? Fencing could be applied to *all* RKEYs on the same QP. This is an area that will need further discussion.
    - With QP scope (single home) with multiple devices (some writes going to memory, some to MAD device) – should/can the single ordering point still be the CPU/IIO complex?
    - With the current implementation, SW can utilize fi\_send\* or fi\_read\* fencing to control ordering of writes to different RKEYs
    - **SNIA NVM TWG Feedback:** We should consider fencing of all rkeys on the same QP.
* **Non-standard memory devices**
  + The high level libfabric and libibverbs API is suggested here but the underlying linkage to the RNIC kernel driver and RNIC needs to be fleshed out.
  + How does the RNIC know which resources and how many additional resources are linked to a particular RKEY?
  + **OFA DS/DA Feedback**: Not clear that FI\_NON\_STANDARD\_MEMORY\_DEVICE indicator should be exposed to the application layer at all. As part of registering the virtual memory for these types of devices, the kernel components can link the additional resources required by the RNIC at the kernel level, encapsulated in the platform provider layer. **Open:** Without this indicator, how will the kernel components know that this region is on a special device that requires attaching of additional resources??
* **Allocating SQ, RQ, and CQ from PMEM**
  + Not addressed in this proposal but there are probably additional complications for recovery and cleanup after power fail if these queues are utilizing pmem
  + **SNIA NVM TWG Feedback:** Don’t allow QPs to be allocated from pmem is a reasonable limitation
* **fi\_mr\_reg**
  + FI\_UNCACHED hint - **OFA DS/DA Feedback**: Consider cache/noncache hint be made available to initiator application. Review SW use cases and gather OSV feedback on whether this indicator should be exposed to the initiator sw. This would require this bit to be put on the wire as part of the fabric extensions.
* **Overloading fi\_writemsg**
  + By adding FI\_COMMIT and FI\_IMMED to existing fi\_writemsg API, it makes it look like we are asking for changes to existing fabric write protocol to add new indicators, which is not the proposal
  + The intent of the proposal is that both these flags would be treated as new fabric opcodes and would NOT affect existing fabric write proposal.
    - fi\_writemsg with FI\_COMMIT would be a new unique opcode on the fabric
    - fi\_writemsg with FI\_COMMIT and FI\_IMMED would be a new unique opcode on the fabric
  + Consider making these new API and not overloading existing API
* **Atomicity guarantees**
  + Consider additional libfabric interfaces for programmatically determining the maximum supported platform atomicity “chunk”. This needs to comprehend the atomicity of each HW component in the data path.
  + Could spec the 8 byte guarantee as part of the fi\_writemsg with FI\_COMMIT but if it changes it is more extensible to make this a SW discoverable attribute of the endpoint in the connection.

Here is a table of the most obvious architectural differences between this proposal and other SNIA driven interfaces:

|  |  |  |  |
| --- | --- | --- | --- |
| **Highlighted differences between proposals** | **Intel libfabric SW API proposal** | **Public SNIA HA White Paper** | **Tom Talpey’s Public IETF Draft Proposal** |
| **Scope of write data to make durable** | -Writes preceding the write with commit and the write commit data are all in the scope of write data to be made durable when sent to the same RKEY representing a pmem registered memory region on same QP  **Implicit Commit List** | -An explicit list of data regions defines the scope of write data to make durable proposed in the “OptimizedFlush” payload. Preceding writes are required to move the data contained in the commit list.  -QP or RKEY limitation is specified (implied)  **Explicit Commit List** | -An explicit SG list of data regions defines the scope of write data to make durable in the “RDMA Commit” payload  -Preceding writes are required to move the data contained in the commit list  -The commit list is the minimum data that must be made persistent and other data written to persistent memory may be committed at any time  **Explicit Commit List** |
| **Controlling write data placement ordering at the target** | -All writes requiring strict data durability ordering require use of commit & fence flag in separate write requests when sent to the same RKEY representing a pmem registered memory region on same QP | -Ordering implied by optimized flush semantics | -Single RDMA Commit operation provides optional 64bit write data to be made durable only after explicit list of data regions have been made durable  See open architecture notes above about atomicity guarantees. |