

# OpenRDMA Project: Building an RDMA Ecosystem for Linux

Venkata Jagana  
Linux Technology Center, IBM  
jagana@us.ibm.com

Bernard Metzler, Fredy Neeser  
Zurich Research Laboratory, IBM  
bmt,nfd@zurich.ibm.com

## Abstract

*The Internet Engineering Task Force (IETF) is defining a set of protocols, also known as the iWARP stack, for Direct Data Placement and Remote Direct Memory Access (RDMA) over IP-based transports. The iWARP stack has the potential to solve the current problem of high host processing overhead for user data movement to and from the network. Nevertheless, its industry-wide acceptance is strongly dependent on the establishment of an appropriate ecosystem, i.e., of an RDMA host software architecture and a corresponding full-featured open source reference implementation for Linux. This paper evaluates architectural alternatives and relates them to the design and choice of the programming interfaces currently being defined by the industry. Focusing on the recently started OpenRDMA project for Linux, it identifies one architecture as particularly attractive due to its multi-vendor support with centralized, vendor-independent resource management.*

## 1 Introduction

The host processing overhead associated with network I/O has been considered a limiting factor for efficient communication over high speed links for more than two decades. For the TCP/IP stack, the payload touching operations (such as data copy/checksum) have been identified as the bottleneck operation during the processing of large packets, while the processing of small packets is mainly limited by protocol stack execution, interrupt processing and context switching [2]. While several attempts have been made to overcome this problem by restructuring the communication stack implementation (for example, protocol offloading, single copy or zero copy stacks, memory mapped I/O and user space protocols), all solutions remained proprietary. The complexity of the proposed solutions together with constant improvements in available hardware performance (CPU, memory and I/O subsystem) limited their acceptance [3].

Recent improvements in networking technology (such as 10Gb-Ethernet and its expected successors) together with a foreseeable break in Moore's Law exacerbate the above *host bottleneck problem*. A well-known solution is to offload the protocol stack onto the adapter *and* to enable zero copies through hardware-assisted mechanisms that place data directly from the adapter into application

buffers and vice versa without involving the host CPU. The IETF is currently standardizing the RDMAP, DDP and MPA protocols [7][1][9], also referred to as the *iWARP stack*, to enable services based on Remote Direct Memory Access (RDMA) over IP-based transports. Based on the well established TCP/IP stack, iWARP thus has the appeal of being applicable on standard media like Ethernet [4]. The significant advantage of this technology is that it provides a high performance, low cost standards-based solution. The iWARP stack is designed to be offloaded together with TCP/IP on an intelligent, RDMA-enabled network interface card, a so called RNIC. Moreover, IETF is defining the necessary semantics that allow fast, asynchronous communication between application and RNIC without any operating system interaction [6].

While the iWARP stack has the potential to solve the host bottleneck problem, it also introduces some problems that need to be solved [3]. These include the development of a standardized OS-to-RDMA interface, the introduction of new, asynchronous and RDMA capable APIs that unveil the stack's full performance to the application, and the integration of the offloaded TCP/IP stack with the host stack for connection endpoint passing, network configuration and management.

## 2 Motivations for OpenRDMA

Currently, several projects are aiming at the integration of dedicated network adapter hardware into the Linux OS to enable a bypass to the kernel transport stack. The most simplistic approach is the integration of so called TCP Offloading Engines (TOE's), which allow for the application transparent offloading of TCP/IP stack processing onto the network interface card. This approach has been discussed controversially in the community; in fact, the expected limited gain in performance seems not to justify the substantial changes to the OS network stack.

### 2.1 Related Projects for the Linux OS

The last years have seen the advent of RDMA capable network technologies like InfiniBand [11], the availability of RDMA network adapters as well as a generic architecture for its host integration [10]. These technologies are accompanied by the specification of vendor independent RDMA APIs like DAPL [12] and IT-

API [13]. Several projects have started to extend the Linux OS incorporating these new technologies. The *Offload Sockets Framework* project aims at the integration of multiple offload technologies into the Linux system, providing transport independent zero copy services over the streaming socket interface [14]. Currently, it supports the use of the Sockets Direct Protocol (SDP) over InfiniBand (IB), TOE like network interfaces as well as IP over Infiniband (IPoIB) as underlying transports.

The *Sourceforge Infiniband Linux Project* was developing a more complete redesign of the communication architecture for Infiniband transport based RDMA support. It delivered an implementation framework for the efficient, vendor independent host adapter integration and the flexible support of multiple RDMA capable APIs including DAPL and IT-API.

Targeting at the same goals, the *OpenIB project* ([www.openib.org](http://www.openib.org)) is aiming to provide open source based implementation of best of breed IB stack for Linux. This project effort is mostly led by adapter vendors, customers and ISVs and the projects developers have been working very closely with the kernel community members. The goal of this project is to have their implementation acceptable to mainline kernel. The OpenIB project community has agreed upon a common, Linux specific programming interface (called VAPI) for IB host channel adapters, but does not support RNICs. The implementation of this interface is currently ongoing.

However, OpenRDMA for Linux is planning to use industry standard RNIC Programming interface to support RNIC devices and provide necessary extensions to support IB. One goal of this project is to work with OpenIB to converge onto a single interface to support both IB and RNICs and thus have a single interface in Linux for RDMA supported media.

## 2.2 Why OpenRDMA for Linux?

The OpenRDMA project intends to provide an open-source implementation of a host software architecture for enabling RDMA services, particularly for iWARP. Although this project is just starting up in the open source world, it is anticipated that this project is likely to be launched before the end of 2004 for the community to participate. The intentions behind the launch of the OpenRDMA project for Linux are the following:

- Both the industry (Systems and RNIC adapter vendors) and the open source community agree upon a common architecture definition of iWARP stack enablement support.
- The creation of an industry (Systems and RNIC adapter vendors) supported common programming interface support for RNIC hardware device drivers.

- The creation of a single common iWARP stack/RDMA service enablement implementation, acceptable to mainline Linux kernel, supported by both the industry (Systems and RNIC adapter vendors) and the open source community.
- Avoiding the mistakes made by other open source projects where there existed multiple open source implementations of the same stack functions.

There are several System and RNIC adapter vendors who have currently agreed on the above intentions and have become part of this OpenRDMA project effort.

## 3 Core Design Issues

The OpenRDMA project is currently discussing alternatives for the RDMA host software, but is quickly converging to a commonly agreed architecture.

An RDMA host software architecture, simply referred to as ‘architecture’ below, enables an OS to provide RDMA services by allowing RNICs to plug into an OS extension that will be referred to as *RDMA Access Layer*, and by integrating iWARP functionality with the kernel TCP/IP stack and OS memory management.

Before a proposed RDMA host software architecture gets introduced in Section 4, important design issues which led to this architecture are to be discussed next.

### 3.1 Evaluation Criteria

**3.1.1 Software Partitioning.** RDMA host software will be partitioned into generic (for Linux also public) OS extensions and vendor-specific components. This partitioning depends on the design and choice of programming interfaces, which can similarly be divided into generic (public for standard interfaces) and Independent Hardware Vendor (IHV) private interfaces.

**3.1.2 RDMA APIs.** The architecture should provide RDMA services to applications through RDMA capable APIs such as the IT-API and DAPL, for both non-privileged (user-space) consumers and privileged (kernel-space) consumers<sup>1</sup>.

**3.1.3 RNIC Interfaces.** For RNIC hardware, multi-device/multi-vendor support should be provided that minimizes code duplication among vendors. RNIC Interfaces (RIs) according to the semantics of the RDMA Verbs [6] (possibly with extensions or modifications) should be utilized, in particular the RNIC-PI currently being specified by the ICSC [15].

---

<sup>1</sup> The term ‘consumer’ will be used to denote any entity making direct calls into a programming interface, regardless of whether that entity is privileged or non-privileged.

Regarding the design of these RNIC interfaces, which separate generic code from vendor-specific code, several options are possible, namely

- RI-1: Public interfaces for kernel-space and user-space consumers,
- RI-2: A public interface for kernel-space consumers and IHV-private interfaces for user-space consumers, and
- RI-3: IHV-private interfaces for both kernel and user-space consumers.

RNIC hardware support must further take into account the link-level (Ethernet) interface, and may support a TCP Offload Engine (TOE) interface.

**3.1.4 Connection Management.** The architecture must provide the connection management functionality expected for the iWARP transport. A core element is the conversion of an established transport connection into RDMA mode, simply referred to as RDMA transition. From the API perspective, the architecture must support both an *immediate RDMA transition* and a *deferred RDMA transition* after exchanging arbitrary streaming mode data. Furthermore, the architecture has to interface with the kernel stack to maintain two entities of the TCP/IP protocol suite, e.g. for migrating the state of a live TCP connection from the kernel stack to the RNIC.

**3.1.5 Event Processing.** A further evaluation criterion is the efficiency of an architecture in demultiplexing work completion events and other RNIC-generated events to OS partitions, kernel clients and user processes, and finally to API notification mechanisms such as the *it\_evd\_wait* call, which allows both waiting on events and reaping the corresponding work completions.

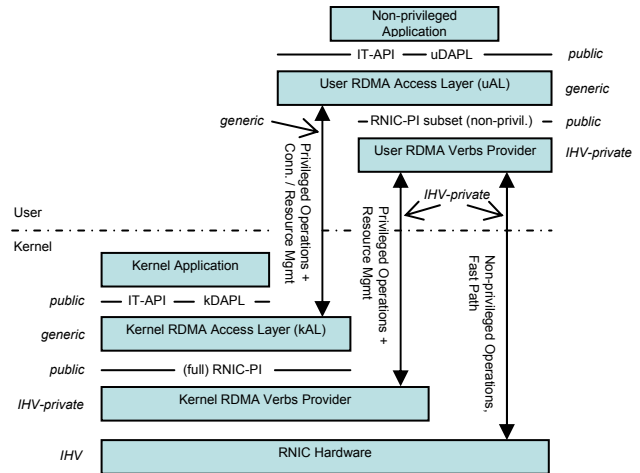
**3.1.6 Memory Management.** The architecture must also have a well-defined interface to OS memory management in order to support the semantics of RDMAP/DDP, and explicit (application-controlled) management of communication buffers for send, receive, and RDMA operations in particular. This interface should enforce *robust pinning and unpinning of communication buffers* and should allow the OS to retain sufficient control over pinned memory, both for system stability and performance. Regarding memory management, it is important to distinguish between communication buffers, which are accessed by both application and RNICs and are typically in system memory, and IHV-private data structures for queues, which may be in system memory or in RNIC memory that has been mapped into host memory.

### 3.2 Two architectural Alternatives

Below, two architectural alternatives based on RI-1 and RI-2 will be compared.

**3.2.1 Architecture A-1.** Figure 1 shows architecture A-1, which is based on option RI-1. Main goals of this architecture are stability and multi-vendor coexistence. The proposed OpenRDMA architecture described in Section 4 is based on A-1.

**Figure 1** Architecture A-1



The OS is extended by a thin, vendor-independent *User RDMA Access Layer* (uAL) in the form of a library and a vendor-independent *Kernel RDMA Access Layer* (kAL) in the form of a kernel module.

Each IHV ships two private software components, namely a *User RDMA Verbs Provider* as a library and a *Kernel RDMA Verbs Provider* that is part of the IHV’s device driver.

The uAL provides the IT-API and/or uDAPL and exploits a subset of the RNIC-PI to IHVs’ user libraries for posting send/receive work requests and for receiving completions, and uses a generic system call interface to the kAL for user requests that require support calls into a privileged and generic OS entity. The use of the generic RNIC-PI thus provides *multi-device and multi-vendor support* since several RDMA user libraries from one or more IHVs can plug into the uAL.

The kAL provides the IT-API and/or kDAPL, as well as the public system call interface with the uAL, and exploits the full RNIC-PI to IHVs’ Kernel RDMA Verbs Providers. Moreover, it interfaces to the kernel TCP/IP stack as well as OS memory management and provides a registration service for IHVs’ Verbs Providers (not shown in Figure 1). The system call interface from uAL to kAL is used for performing privileged operations. In this mode, the kAL either acts as a proxy to perform an operation on behalf of the uAL or provides support functionality that is only available in kernel mode. For instance, OS-controlled pinning and unpinning of communication buffers can be enforced. The feasibility of a centralized, vendor-

independent resource management is a key advantage of A-1. A more detailed description of resource management for A-1 is provided in 4.1.4.

The kAL provides all connection management functionality for kernel clients, while the *uAL in concert with the kAL* provides the same functionality for user-mode applications. A significant advantage of architecture A-1 is that the kAL provides a *single, generic interface to the kernel networking stack*, which is used in particular for TCP context migration. Hence, any API call in user space for establishing an RDMA-enabled connection, e.g., *it\_ep\_connect*, will cause a system call from uAL to kAL for migrating the TCP context, unless the connection was already initiated on the RNIC. Transitioning a user-space Queue Pair (QP) to RTS state is an example that requires a support call into the kAL for completing the operation.

The User RDMA Verbs Provider, also referred to as the *RDMA user library* for brevity, exports non-privileged RNIC-PI calls only, has a private interface for posting work queue elements (WQEs) to the RNIC HW as well as for receiving completion queue elements (CQEs) from the RNIC HW, and another private interface to the Kernel RDMA Verbs Provider for the management of IHV-private resources. Due to the system call interface from uAL to kAL, the RDMA user library is not required to export the full RNIC-PI, i.e., a smaller subset of these is sufficient. In any case, the uAL exploits only that smaller subset.

The Kernel RDMA Verbs Provider exports all RNIC-PI calls and provides the private interface to the User RDMA Verbs Provider. It manages IHV-private data structures for send, receive, and completion queues, both for queues associated with user processes and for queues associated with kernel clients. Furthermore, it manages IHV-private data structures for event queues.

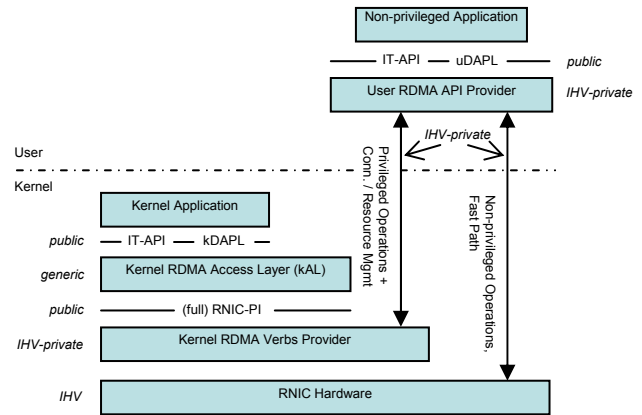
For user-level applications, the fast path for posting send and receive work requests directly to the HW, and for receiving completions from the HW is formed by the User RDMA Access Layer (uAL) and the IHV's RDMA user library.

Applications are expected to access RDMA services only through RDMA enabled APIs such as the IT-API and DAPL. A significant motivation for application writers to program against the IT-API, rather than directly against the RNIC-PI, is that IT-API implementations will handle all iWARP connection management under the covers, for both immediate and deferred transition to RDMA mode. Another motivation to program against the IT-API or DAPL is the availability of transport-independent calls like *it\_ep\_connect*, *it\_listen\_create* and *it\_ep\_accept*, which are supported on iWARP and InfiniBand. Moreover, the architecture is designed such

that uAL and kAL can provide plug-in support for both iWARP and IB RDMA user libraries.

**3.2.2 Architecture A-2.** An alternative architecture based on option RI-2 is depicted in Figure 2.

**Figure 2** Architecture A-2



Here, the OS is extended by a vendor-independent kAL, while a uAL (and the system call interface to the kAL) does not exist.

Each IHV ships two private software components, namely a *User RDMA API Provider* (including API implementation and user RNIC driver functionality) as a library and a *Kernel RDMA Verbs Provider* that is part of the IHV's device driver.

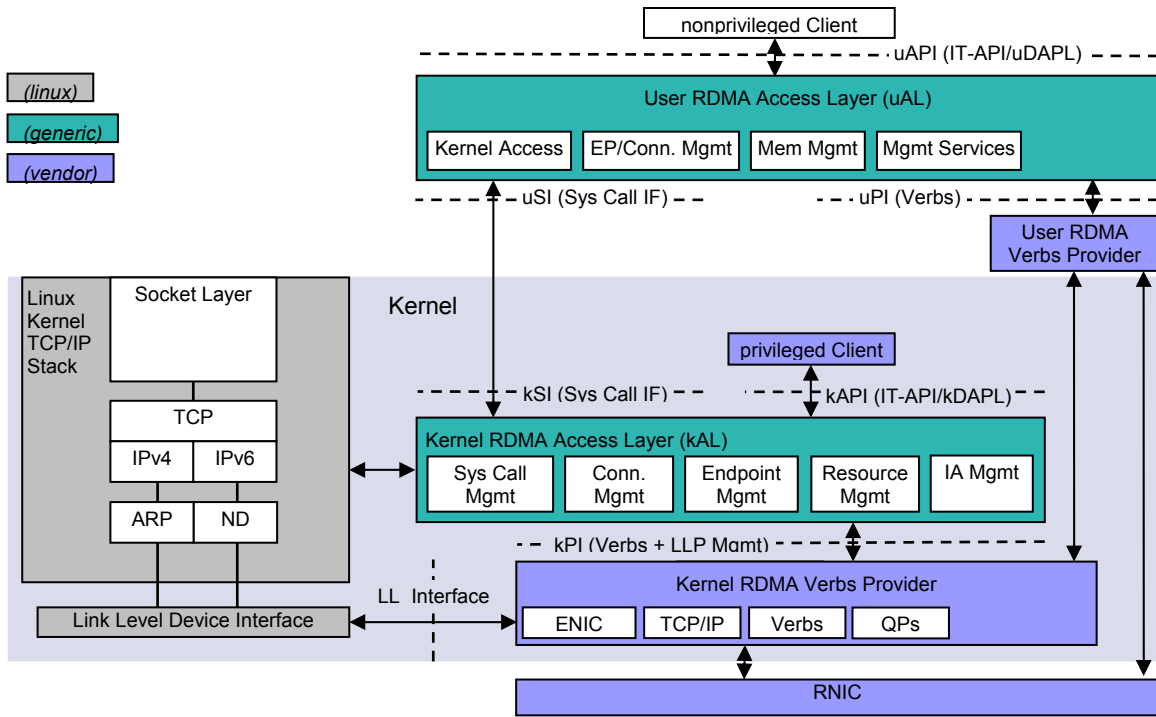
The kAL provides the IT-API and/or kDAPL. It exploits the full RNIC-PI to IHVs' Kernel RDMA Verbs Providers, interfaces to the kernel TCP/IP stack for TCP context migration, and provides a registration service for IHVs' Verbs Providers (not shown in Figure 2). Kernel clients are thus supported in the same way as with architecture A-1.

The User RDMA API Provider directly implements the IT-API and/or uDAPL, has a private interface to the RNIC HW for the fast path, and another private interface to the Kernel RDMA Verbs Provider for all resource management. As a result, multi-vendor support requires multiple IT-API or uDAPL implementations.

The Kernel RDMA Verbs Provider exports all RNIC-PI calls and the private interface to the user library. It manages IHV-private data structures for queues. Furthermore, it also controls the (un)pinning of communication buffers.

Therefore, if the RNIC-PI exposes only the memory registration semantics of the RDMA Verbs, then multiple (de)registrations of the same or overlapping communication buffer(s) with RNICs from different vendors are not safe. It is possible to extend the RNIC-PI with additional upcalls to the kAL for (un)pinning a

**Figure 3** Proposed OpenRDMA Architecture



virtual memory area. Adding such upcalls can provide safe (un)pinning, if the upcalls perform reference counting for pinned-down pages. However, the *presence of the upcalls does not force IHVs to actually use them*. If some IHV uses other OS-provided methods to (un)pin communication buffers, system integrity may be compromised. Moreover, designing OS mechanisms for limiting or reducing the amount of pinned memory is potentially more difficult with IHV-controlled (un)pinning.

Another drawback of A-2 is the fact that both the OS and the IHVs must provide IT-API and/or DAPL implementations, increasing code maintenance costs. Furthermore, a consistent view of user and kernel API resources such as protection zones and endpoints for the management of the RDMA stack will not be possible, particularly in the multi-vendor case. Note that one could modify A-2 by collapsing the kAL and the Kernel RDMA Verbs Provider into a single, IHV-private Kernel API Provider. This would remove the RNIC-PI entirely and lead to a single IT-API or DAPL implementation (with one IHV). This approach would cause even more code duplication among IHVs.

## 4 Proposed OpenRDMA Architecture

Figure 3 gives an overview of the proposed host software architecture for the OpenRDMA stack. As can be seen, it closely follows the architecture introduced as A-1. It distinguishes generic extensions to the Linux system and RNIC vendor-specific functionality. The architecture provides RDMA services to user and kernel level applications. It also interacts with the RNIC(s) through the RNIC-PI, with the Linux Kernel TCP/IP stack, and with OS memory management.

Main building blocks are a *Kernel RDMA Access Layer*, a *User RDMA Access Layer*, and *User and Kernel RDMA Verbs Providers*. For both unprivileged (user level) and privileged (kernel resident) consumers, the architecture supports protected direct access to the RNIC HW for the efficient implementation of the fast path.

### 4.1 Kernel RDMA Access Layer

The Kernel RDMA Access Layer (kAL) provides IHV-independent OS support for RDMA operations. It directly supports kernel-level consumers and also privileged operations of user-level consumers. The kAL supports the multiplexing of several RNICs. The kAL employs OS-specific interfaces with the kernel TCP/IP stack and OS resource management. It offers generic interfaces to the User RDMA Access Layer, to the

Kernel RDMA Verbs Provider(s), and to kernel clients such as NFSv4 over TCP. It is responsible for opening and closing RNICs on behalf of consumers. The kAL is composed of some framework functionality and function-specific building blocks, which are introduced next.

**4.1.1 System Call Management.** This function manages the execution of privileged operations requested by the uAL's kernel access function via ioctls. It interacts with other kAL functions to perform the requested operation. The system call interface is used to support the following operations when performed by user-mode consumers:

- Protection Domain (PD) management
- (De)registration of communication buffers (MRs)
- Queue Pair (QP), Completion Queue (CQ) and Shared Receive Queue (SRQ) management
- Connection management

The system call interface is also used to pass events to event dispatchers in user space.

**4.1.2 Connection Management.** The Connection Management (CM) is responsible for both the management of Lower-Layer Protocol (LLP) connections associated with RDMA services and the integration of the offloaded LLP stack with the kernel stack. Currently, LLP support is limited to TCP/IP, but could be extended to SCTP. The CM does not implement an LLP stack on its own; rather it interacts with both the kernel stack and the offloaded stack to support the needed functionality.

The CM is responsible for connection setup, context migration, and takedown. It can use kernel sockets to set up TCP connections and also accepts user-established TCP endpoints (user-level sockets). The CM uses a portable representation of a local TCP context to be moved to an offloaded TCP stack implementation during RDMA transition. It supports both immediate and deferred RDMA transition.

The ICSC IT-API workgroup is currently specifying a new transport-dependent interface for iWARP to support the deferred RDMA transition. This interface allows the seamless conversion of an existing kernel TCP/IP connection to an offloaded RDMAP connection after exchanging an arbitrary amount of streaming mode data. The new interface is required, for instance, by the iSCSI Enhancements for RDMA (iSER) protocol [16]. In addition, IT-API Issue 1 already provides a transport-independent interface based on *it\_ep\_connect*, *it\_lp\_create*, and *it\_ep\_accept*, which can be implemented on iWARP through an immediate RDMA transition, after establishing an implicit transport connection under the API covers. Both of these connection management interfaces require the generation and parsing of the MPA

Request/Reply messages<sup>2</sup> within CM. CM supports both interfaces for kernel clients. For user-space applications, the immediate RDMA transition can be implemented using the kAL CM functionality, while the deferred RDMA transition must be implemented with uAL connection management support.

CM further synchronizes the coexistence of host resident LLP stack and offloaded stack. This includes port space management to ensure that a local port gets bound only once. Moreover, the CM can provide host global information for IP routing and link-level address resolution in a generic format, which can be downloaded to the RNIC through an LLP Management extension of the RNIC-PI. Similarly, MIB statistics collected by the offloaded stack can be passed to the kernel stack in order to provide a consistent view for SNMP-based stack management.

#### **4.1.3 Endpoint Management and Event Handling.**

*Endpoint* (EP) management for APIs is responsible for managing the endpoint state machine. Each API defines such an EP state machine and the set of allowed transitions for each state. Transitions can be triggered by consumer actions, LLP events, or RNIC interface events. At the API level, each EP has associated *event dispatchers*. The API implementation establishes a one-to-one correspondence between an EP and an underlying QP, and between an event dispatcher and an underlying CQ.

While EP management synchronizes EP state with QP-related events, there is no direct correspondence between QP and EP states. QP management itself (QP setup, WR insertion and deletion) is not performed by the kAL, but implemented by the RDMA Verbs Provider. EP management interacts with the CM function for connection setup and context migration (see above).

For each event dispatcher and corresponding CQ, the kAL event handling maintains a separate event queue, regardless of whether the CQ resides in the kernel or in a user process context. Since an RNIC may provide fewer Completion Event Handlers (CEHs) than CQs [6], supporting multiple CQs enforces SW-based event demultiplexing.

Event handling registers kAL-provided CEH(s) with the RNIC-PI. At CQ creation time, the CQ gets associated with one of these event handlers. When a work request completes, the RNIC places a CQE into the associated CQ and notifies the kAL by raising an interrupt, which causes the execution of the associated CEH. The CEH demultiplexes the completion event to the appropriate event queue.

---

<sup>2</sup> The RDMAP Verbs [6] imply that the MPA Startup messages have to be processed above the RNIC-PI.

A kernel client can be notified by the CEH via a callback function associated with the CQ or by waking up a kernel thread. In case of a user-space application waiting on an event dispatcher, the CEH also places a wakeup on the Sys Call interface.

**4.1.4 Resource Management.** This function provides unified resource management (RM) for all attached RNICs. In this section, ‘RNIC’ will refer to a *virtual RNIC* or, equivalently, an *RNIC handle* [6], unless identified explicitly as a *physical RNIC*. The RM abstracts RNICs by *API Interface Adapters* (IAs).

In particular, the RM provides generic management of communication buffers, referred to as *Local Memory Regions* (LMRs) in [12][13] and as *Memory Regions* (MRs) in [6]. It interfaces with OS memory management for translating virtual addresses to physical addresses and to perform (un)pinning of physical memory pages. API requests from user space like *it\_lmr\_create* or *dat\_lmr\_create* for creating memory regions are passed from each uAL to the kAL, which asks the OS to perform translation and pinning. The resulting Physical Buffer List (PBL) and the parameters of the virtual memory area are then passed to the Register Non-Shared Memory Region verb.

The generic kAL (and thus the OS) gains a complete view of pinned communication buffers and control over (un)pinning operations, which avoids IHV-specific, possibly uncoordinated interactions with OS memory management. This leads to safe (un)pinning of communication buffers, even in case of *multiple (de)registrations* [6] of the same or overlapping buffer(s) with RNICs from different IHVs. Multiple registrations can result from sharing virtual memory areas across user processes or different RNICs.

Such OS-controlled (un)pinning has the added benefit that the OS can employ safety mechanisms that limit the amount of pinned memory requested by applications. As a further extension, it may facilitate the design of an OS mechanism for reclaiming pinned pages on demand, e.g., by asking API consumers to invalidate memory regions through callbacks, where candidate memory regions could be selected based on an LRU scheme, priorities, etc.

A main responsibility of RM is the bookkeeping of both API objects and corresponding RNIC resources. All consumer requests requiring RNIC or host resources are tracked by the RM. The RM employs *system-unique handles* for referring to most kAPI objects, which are exposed at the Kernel API. Moreover, the RM provides a one-to-one mapping between the system-unique handles and pairs comprising an IA handle and an RNIC-provided, *locally unique handle or ID*. Note that RNIC-provided handles and IDs are in general not unique across

RNICs. For example, EP handles are system unique, while QP handles are locally unique.

API objects and RNIC resources can be organized in different ways. One possibility is to start from the hierarchical structure of API objects (IA, PZ, EP, SRQ, LMR, RMR), which represent resources (RNIC, PD, QP, SRQ, MR, MW) of one or more physical RNICs. Another possibility is to consider (separately for each RNIC) the hierarchical structure of resources (PD, QP, MR, MW), which are associated with API objects. The former is more useful for accessing API objects, while the latter is more useful for per-RNIC resource management. Knowing the hierarchical structure of API objects or RNIC resources allows robust clean-up of all child objects and resources belonging to the same released parent (for example, an RNIC PD might contain several QPs, a SRQ and a CQ).

**4.1.5 Interface Adapter Management.** This function is responsible for loading and unloading the IHV-specific Kernel RDMA Verbs Provider and connecting it with the kAL. Furthermore, it lists available interfaces upon the consumer’s request, which allows the consumer to create an IA in a second step. The function interacts with RM to support graceful resource de-allocation in case of a failing RNIC.

## 4.2 User RDMA Access Layer

The User RDMA Access Layer (uAL) provides functionality similar to the kAL. While all non-privileged operations are passed to the User RDMA Verbs Provider, privileged operations are forwarded to the kAL via the uAL’s kernel access function. If no user-level RDMA Verbs Provider exists, then all operations can be mapped onto functionality provided by the kAL and its associated Kernel RDMA Verbs Provider.

As mentioned previously, connection management functionality for user-mode applications is provided by the uAL in concert with the kAL.

The uAL can be used to implement Aggregate Event Dispatchers (AEVDs) for aggregating completions from Simple Event Dispatchers (SEVDs). Implementing AEVDs in the uAL allows aggregating completions from CQs associated with different IAs.

## 4.3 RDMA Verbs Provider

It is expected that every RNIC vendor will provide a kernel-level RDMA Verbs Provider module. Optionally, it may provide a user-level RDMA Verbs Provider library for efficient user-level RDMA services. RDMA Verbs Providers encapsulate functions that are private to the RNIC implementation, such as QP management and conversion of work requests to WQEs.

**4.3.1 Kernel RDMA Verbs Provider.** This module provides vendor-specific software implementing the RDMAP verbs. It translates the RDMA verbs presented by the kAL into appropriate actions such as the creation and management of QPs and the insertion and removal of WQEs. All QP, CQ and SRQ data structures are under direct control of the module and are not accessible to the consumer.

Additional code is necessary to provide generic Ethernet driver functionality for non-offloaded connections and for the forwarding of TCP/IP stack control information such as the ARP table, routing table or MIB informations.

While the module's interfaces to the User RDMA Verbs Provider and the NIC hardware are IHV-private, its upper interface to the kAL will implement the RNIC-PI currently being defined within the Open Group.

**4.3.2 User RDMA Verbs Provider.** This library contains vendor-specific software for providing user-accessible RDMAP Verbs services and for direct access to the NIC hardware. It establishes a fast path for all performance critical operations that can be performed without holding exclusive access permissions. It includes all send and receive type operations. As with the Kernel RDMA Verbs Provider, all QP, CQ and SRQ data structures are under direct control of the library and are not accessible to the consumer.

In addition to the RNIC-PI with the uAL, the library will implement a private interface with the Kernel RDMA Verbs Provider of the same RNIC. This interface is needed for private RNIC resource management, such as the mapping of doorbell registers and the creation of QPs.

The User RDMA Verbs Provider will further implement private RNIC interfaces for the signaling and processing of WQEs and CQEs, thus establishing the fast path between consumer and RNIC hardware.

## 5 Completing the OpenRDMA Ecosystem

In order to provide standards based RDMA services on Linux, it is necessary to build the overall ecosystem for the RDMA host software architecture. This ecosystem consists of several components, including

- Standardized RDMA API support for use by middleware and applications.
- Standard RNIC Verbs programming interface (RNIC-PI) for separating generic code from IHV-provided code.
- Establishing a process to ensure conformance with the iWARP stack and interoperability.

There are various activities within the industry contributing to such an ecosystem. The OpenRDMA

project will continue to track these activities and participate as necessary:

**5.1.1 RDMA API Support.** OpenRDMA will support both DAPL and the IT-API, which are evolving to support iWARP-based RDMA services. It is currently not the intention of this project to support service access via the existing socket interface.

The current plan is to incorporate the API functionality into the kAL resp. the uAL. In this way, kAL and uAL will directly export the APIs to the RDMA service consumers. For the sake of clear modularization, it is debatable whether the kAL/uAL should instead export a generic RDMA service access interface, on top of which different API implementations could be placed.

**5.1.2 RNIC-PI Specification.** The RDMA Verbs Specification provides a semantic description of an interface offered by the RNIC adapters. However, there is currently no syntactical definition of this interface.

As already mentioned, there is an ongoing industry effort for defining a programming interface for RNIC Verbs, known as the RNIC Programming Interface (RNIC-PI). This effort is addressed by the RNIC-PI workgroup within The Open Group/ICSC [15]. Rather than creating its own RNIC programming interface, the OpenRDMA project relies upon the RNIC-PI. Because the specification of the RNIC-PI is still ongoing, the OpenRDMA project will track its progress and update a corresponding reference implementation to achieve a stable interface definition.

**5.1.3 iWARP Conformance and Interoperability.** One of the goals of the OpenRDMA project is to provide a standards conformant and industry interoperable Linux implementation for iWARP stack/RDMA service enablement. In order to ensure the conformance and interoperability of this implementation, the project's intent is to participate in the industry's interoperability events very early on in the development process.

## 6 Conclusions

There has been considerable interest shown by several system and adapter vendors to participate and contribute to this open-source based OpenRDMA effort. This effort is currently moving in the right direction - to get the vendors and community to agree upon a single architecture and an open-source implementation acceptable for the Linux mainline kernel.

### 6.1 Call for Joining the OpenRDMA Effort

Since OpenRDMA will be a fully open sourced project, any individual developer, user, or vendor having a specific interest in iWARP host software development for

Linux can contribute to this project. In fact, the contribution(s) can be made in different ways, e.g.,

- through new code development for a feature enhancement,
- by testing the code as a user and informing the community about the problems, or
- by fixing bugs and submitting patches to the project.

A separate email reflector is set up to discuss and agree upon the architecture before the project will be made fully open to the general developer and user community. The current system and adapter vendor participants in this effort have agreed that this project will be completely open, and will thus follow the Linux open source project de-facto guidelines. The OpenRDMA project is consensus-driven and, although supported by several vendors, not controlled by any individual vendor. The OpenRDMA project is hereby requesting the vendors/developers/users to join and contribute to the OpenRDMA effort.

## References

- [1] S. Bailey, T. Talpey, The Architecture of Direct Data Placement (DDP) and Remote Direct Memory Access (RDMA) on Internet Protocols, draft-ietf-rddp-arch-05.txt, 2004-7-13
- [2] D. Clark et al., An Analysis of TCP Processing Overhead, IEEE Communications Magazine Vol. 27, June 1989
- [3] J. Mogul, TCP offload is a dumb idea whose time has come, 9th USENIX Workshop on Hot Topics in Operating Systems, May 2003
- [4] A. Romanow, S. Bailey, An Overview of RDMA over IP, PFLDnet 2003 - First International Workshop on Protocols for Fast Long-Distance Networks, February 2003
- [5] A. Romanow et al., Remote Direct Memory Access (RDMA) over IP Problem Statement, draft-ietf-rddp-problem-statement-04.txt, 2004- 7-13
- [6] J. Hilland et al., "RDMA Protocol Verbs Specification", draft-hilland-rddp-verbs-00.
- [7] R. Recio et al., "An RDMA Protocol Specification", draft-ietf-rdmap-03.txt.
- [8] H. Shah et al., "Direct Data Placement over Reliable Transports", draft-ietf-rddp-ddp-02.txt.
- [9] P. Culley et al., "Marker PDU Aligned Framing for TCP Specification", draft-ietf-rddp-mpa-01.txt.
- [10] Compaq, Intel, Microsoft. VI Architecture Specification V1.0, December 1997
- [11] InfiniBand Architecture Specification Release 1.0a, downloadable at <http://www.infinibandta.org/specs>
- [12] Direct Access Transport APIs. uDAPL online at <http://www.datcollaborative.org/udapl.html> and kDAPL online at <http://www.datcollaborative.org/kdapl.html>
- [13] The Open Group. Interconnect Transport API (IT-API), Issue 1.0, February 2004,
- [14] Architectural Specification – Offload Sockets Framework and Sockets Direct Protocol (SDP), online available at [http://infiniband.sourceforge.net/archive/LinuxSAS\\_SDP.pdf](http://infiniband.sourceforge.net/archive/LinuxSAS_SDP.pdf)
- [15] The Open Group. RNIC Programming Interface Working Group, work in progress, <http://www.opengroup.org/icsc/rnicpi/>
- [16] M. Ko et al., iSCSI Extensions for RDMA Specification, draft-ko-iwarp-iser-02.txt, July 2004

## Legal Statement

This document represents the views of the authors and does not necessarily represent the view of IBM. IBM is a registered trademark of International Business Machines Corporation in the United States, other countries or both. Other Companies products may be trademarks of others.