Memory Management Support for Multi-Programmed Remote Direct Memory Access (RDMA) Systems

Kostas Magoutis
IBM T. J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
magoutis@us.ibm.com

Abstract

Current operating systems offer basic support for network interface controllers (NICs) supporting remote direct memory access (RDMA). Such support typically consists of a device driver responsible for configuring communication channels between the device and user-level processes but not involved in data transfer. Unlike standard NICs, RDMA-capable devices incorporate significant memory resources for address translation purposes. In a multi-programmed operating system (OS) environment, these memory resources must be efficiently shareable by multiple processes. For such sharing to occur in a fair manner, the OS and the device must cooperate to arbitrate access to NIC memory, similar to the way CPUs and OSes cooperate to arbitrate access to translation lookaside buffers (TLBs) or physical memory. A problem with this approach is that today's RDMA NICs are not integrated into the functions provided by OS memory management systems. As a result, RDMA NIC hardware resources are often monopolized by a single application. In this paper, I propose two practical mechanisms to address this problem: (a) Use of RDMA only in kernel-resident I/O subsystems, transparent to user-level software; (b) An extended registration API and a kernel upcall mechanism delivering NIC TLB entry replacement notifications to user-level libraries. Both options are designed to re-instate the multiprogramming principles that are violated in early commercial RDMA systems.

1. Introduction

The need to reduce networking overhead in system-area networks in the early 1990's motivated a flurry of research on user-level networking protocols. Projects such as SHRIMP [2], Hamlyn [4], U-Net [23] and others, proposed user-level access to a network interface controller (NIC) as an approach that offers two primary benefits: First, it enables host-based implementations of new, lightweight networking protocols with lower overhead compared to kernel-based TCP/IP protocol stacks. Second, for applications requiring use of the TCP/IP protocol stack, there is a potential for application-specific customization of user-level libraries. In addition to the ability for user-level access to the NIC, which is the defining feature of user-level NICs, most of these projects also advocated a new host programming interface to the NIC. This programming interface is based on a *queue-pair (QP) abstraction* and requires *pre-posting of receive buffers* [1] [22]. It is important to note that this programming interface is *not* a defining feature of user-level NICs and can be implemented without special NIC support [17]. In contrast, user-level access to the NIC necessarily requires special NIC support, which increases the complexity of NIC design as described later in this paper.

A feature of the networking API introduced by user-level NICs is the ability for remote direct data placement (RDDP), *i.e.*, direct data transfer between the network and the target memory buffers without any intermediate data movement. The key benefit of RDDP is elimination of unnecessary memory system and CPU overhead incurred in systems built over standard TCP/IP interfaces and traditional NICs [13]. RDDP is possible with either *unsolicited* or *solicited* I/O operations. In the former case, incoming network I/O data are deposited in anonymous buffers posted prior to the I/O taking place. In the latter case, explicit *tags* are used (and carried on the incoming data headers) to identify the user-level buffer that is the target of the I/O operation. Tags for solicited RDDP can take various forms [12]; one widely used option is using the virtual memory address of user-level buffers as their RDDP tag. This flavor of RDDP is known as remote direct memory access (RDMA).

The benefits of user-level RDMA-capable network interface controllers (RNICs) include lower CPU overhead and flexibility due to the bypassing of the kernel. The overhead reduction is partly due to transport protocol offload[1], the avoidance of unnecessary data movement via memory copying, and the avoidance of the

[1] RDDP requires total or partial transport protocol offload, particularly when TCP/IP is used as a transport, since RDDP operates at a higher level in the networking stack [16].

user-kernel boundary crossing. The bypassing of the kernel makes possible user-level implementations that are customized for applications and also the avoidance of buggy kernel components [23]. An additional benefit of the networking API is asynchrony without necessarily requiring OS support for it [13].

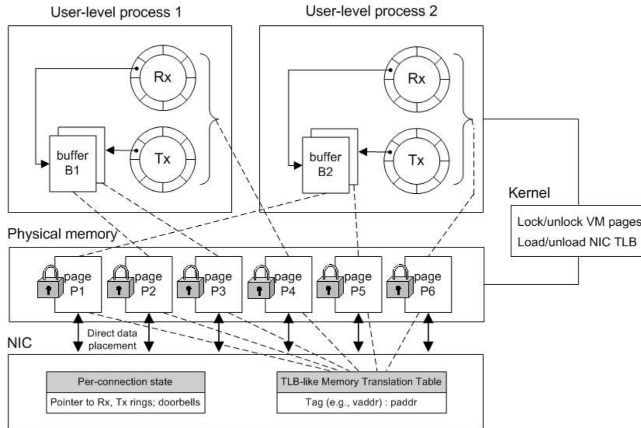

Figure 1. **Two processes sharing a user-level RNIC.**

User-level RNICs often involve complex system architectures (Figure 1). The programming interface that user-level networking libraries use to control such RNICs typically consists of a pair of receive (Rx) and transmit (Tx) rings, mapped in the address space of each communicating process (typically managed by a user-level library) and shared between the process and the RNIC. User-space buffers used for communication must be described in terms of their virtual memory addresses. Since the RNIC must be able to resolve such virtual addresses into physical (bus) addresses to initiate direct memory access (DMA) operations, an RNIC typically includes a TLB-like address translation structure.

Most commercially available RNICs today are not integrated into host and OS memory systems. This is due to two main reasons: First, most OSes do not provide support for device-specific page tables [20]; the alternative of incorporating such abstractions in device drivers involves significant complexity. Second, most RNICs do not offer any mechanisms for handling TLB miss faults, which would require suspending the faulting RDMA operation, throwing an exception that can be handled by the OS to load the missing translation (and possibly servicing a virtual memory (VM) page fault from disk), and restarting the operation [23]. This inability of RNICs to support *on-demand loading* and *unloading* of address translations means that these activities must instead happen *proactively*, *i.e.*, a process (or the kernel) must load a translation to the NIC TLB prior to the translation being used and (voluntarily) unload the translation at some point in the future. The process of adding address translations to a user-level RNIC is currently performed by user processes using a `REGISTER` system call. Similarly, address translations may be voluntarily removed from the RNIC TLB using a `DEREGISTER` system call. Registration typically also involves pinning the VM pages whose address translations are loaded on the RNIC TLB in physical memory.

The inability to support on-demand loading and unloading of RNIC TLB entries rules out the option of using host memory as a second-level address translation structure to extend the RNIC addressing capabilities. The only other practical option that can meet the requirements of I/O-intensive applications is to equip RNICs with large TLBs, which is standard practice today. This option however, increases the cost of the RNIC and does not work well in multiprogramming environments where the RNIC is shared by multiple applications. New designs that reduce RNIC TLB size requirements, better utilize those TLBs, *and are practical to implement* in commercial RNICs, could be a major driving force towards wider adoption and deployment of RNICs. Two possible ways to achieve these objectives are:

1. Use NIC address translation resources only for operations that inherently require their use; such operations do not include messaging (*i.e.*, unsolicited RDDP) nor control operations, *e.g.*, access to control data structures for the purpose of communicating between the NIC and user-level networking libraries.

2. Increase the degree of utilization and avoid hoarding of the NIC TLB in multiprogramming environments, where the RNIC is shared by multiple processes. With current RNICs, it is possible that a greedy process allocates a large chunk of the RNIC TLB without actually using all of it. This requires the ability to forcibly unload translations from the NIC TLB.

In this paper we present two design options that can achieve the above goals. The first option is to use RDMA only in kernel-resident I/O subsystems, transparently to user-level software layers. This approach, which assumes that RNICs are only accessible via a kernel host interface, requires fewer RNIC TLB resources and allows for efficient and fair global policies in sharing the RNIC TLB. We describe such a design in Section 2. The second option is to provide a collaborative mechanism between user-level RNICs and user-level I/O libraries, whereby the kernel notifies the library with an upcall when a TLB entry must be de-registered; subsequently, the user-level library must take appropriate action to decommission the affected communication buffer(s). This mechanism, which addresses point (2) above, is described in more detail in Section 3.

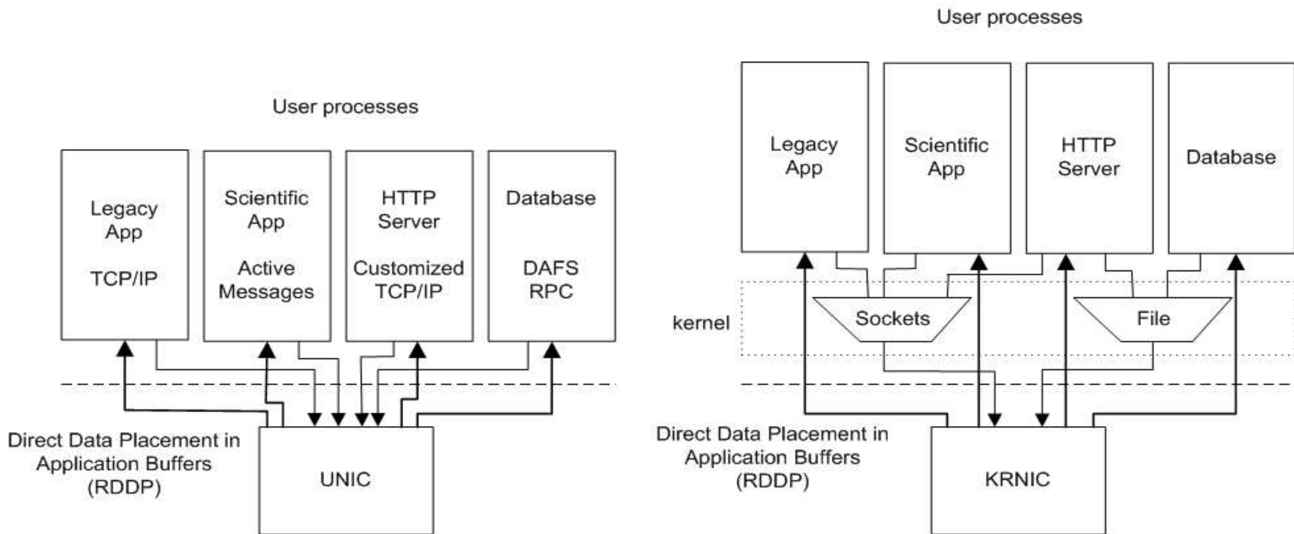

Figure 2. User-level vs. kernel-based RNIC. RDDP is possible in both cases, if supported by the I/O API [5][10] [13].

2. Kernel-based RDMA

Kernel-based RNICs are defined to be RNICs that are accessible to hosts *only* via a kernel interface (Figure 2-right). As such, they are used by kernel-based I/O subsystems (*e.g.*, network file systems [5] and storage device drivers [6]), whereas user-level RNICs can be used by either user-level or kernel-based I/O subsystems [13]. In previous work we argued that the performance drawback of using a kernel-based RNIC instead of a user-level RNIC amounts to the overhead of crossing the user-kernel boundary for issuing and managing I/O operations, which is not a significant cost for I/O-intensive network storage workloads [14]. In this paper, we argue that kernel-based RNICs lend themselves to simpler and more efficient system designs than user-level RNICs. In particular, on the issue of efficient use of RNIC memory resources, user-level RNICs have at least two fundamental drawbacks compared to kernel-based RNICs. First, user-level RNICs inherently use more memory resources for address translation purposes. Second, user-level networking systems do not currently offer the necessary controls to enforce efficient and fair sharing of RNIC TLB resources in multiprogramming environments.

In more detail, user-level RNICs store virtual address translations and access control information on the RNIC TLB for three main purposes: (a) control structures, such as queues and transfer descriptors; (b) messaging buffers; and (c) RDMA buffers. However, the need to use the RNIC TLB for (a) and (b) can be eliminated in kernel-based RNICs. First, control data structures and messaging buffers need not be mapped in user address space and can thus be referenced by physical address only. Since the kernel is trusted to enforce the proper access control in the case of control data structures and messaging buffers, there is no need to dedicate resources for that purpose on the RNIC. This, however, is not true for RDMA buffers since memory accesses can be initiated by an untrusted party and thus the access rights of the RDMA tags must always be verified.

Another drawback of current commercially-available user-level RNIC systems is their lack of mechanisms to control the consumption of RNIC TLB resources by individual processes. User-level I/O libraries typically try to register with the RNIC as much of their communication buffer pool as possible. This is because user-level processes lack the incentive to be "good citizens" and to act unselfishly by de-registering buffers or by performing per-I/O registration, which would voluntarily reduce their RNIC TLB usage. Thus, greedy processes can monopolize the RNIC TLB, whether they use their TLB allocations or not, and refuse access to other processes. Although it is in principle possible to place limits on the maximum amount of registration a process can perform (similarly to the MLOCK kernel interface), such limits are typically static (*e.g.*, a pre-set maximum number of TLB entries per process) and can result in underutilization of the NIC TLB.

Stated differently, the issue has to do with the ability to specify and enforce global policy that fairly and efficiently arbitrates sharing of the NIC TLB (and of the physical memory referenced by it) between processes. Ideally, it should be possible to avoid processes selfishly hoarding these resources but also let processes use more resources than their "fair share" if others do not utilize their resources. This problem (the fact that it is impossible to enforce global system policy without some degree of kernel involvement) is *inherent to any user-level system*

designed to bypass the kernel. Another example is Exokernel [11], a system that advocates extermination of all kernel abstractions.

Kernel-based RNICs enable solutions to the above problems. First, regulation of the use of the RNIC TLB by the kernel ensures that each process uses entries in the RNIC TLB for only as long as necessary. This is possible because, in kernel implementations of I/O subsystems, the kernel is aware of the extent of the time interval a memory buffer translation must be known to the NIC and can be trusted to de-register the buffer after that. For example, in many I/O subsystems [5] [13], an RDMA operation is preceded by an RPC request and followed by an RPC response. These two RPC messages designate an upper bound on the duration of the RDMA operation. Thus, per-I/O registration is enforceable, minimizing TLB space use, *i.e.*, a TLB entry is consumed by a buffer only if an RDMA to that buffer is expected soon.

Another way to deal with hoarding processes in kernel-based RNIC systems is to reduce their share of the TLB when necessary. For example, in network or file access protocols that use window-based flow control, the size of the outstanding I/O window in conjunction with the average I/O size give an estimate of the RNIC TLB consumption over a certain network connection. At times, a process with high throughput requirements may be exceeding its “fair share” of NIC TLB entries. In such cases, the kernel can decide to reduce the process’ share of the TLB and communicate that decision to the responsible kernel-resident I/O subsystem. The latter can effect the change by shrinking the appropriate outstanding request window(s), effectively limiting the maximum throughput achievable by that process, and de-registering the appropriate amount of buffers. Revocation of resources, just as described here, is straightforward when performed in the kernel. Similar functionality applicable to RNIC resource consumption by user-level I/O libraries requires additional support and is the subject of Section 3.

In summary, use of kernel-based RNIC by kernel-resident I/O subsystems only, offers the following benefits:

1. The RNIC TLB is used only for essential operations (*i.e.*, RDMA); messaging and control operations do not require use of the TLB when implemented in the kernel.
2. The kernel can ensure economical consumption of RNIC resources on behalf of processes; for example, per-I/O registration and de-registration of buffers can be enforced.
3. Besides economy of use, at times of contention for RNIC TLB resources, the kernel can further yield resources used by certain processes if necessary.

Point (1) is inherent to kernel-based RNICs. Points (2) and (3) are matters of global policy, which is naturally implementable in the kernel but can also be applied to user-level systems with appropriate mechanisms, as discussed in the next section.

3. A Collaborative Upcall-based Protocol

As argued in Section 2, there is currently a lack of mechanisms to ensure that selfish user-level processes that over-consume RNIC resources are forced to yield some of those resources at times of contention. In this section, we present a protocol that requires the collaboration of user-level RNICs, the kernel, and user-level I/O libraries to enable the revocation of unused or unlikely-to-be-used-soon RNIC TLB entries from certain processes and their re-allocation to others which have an immediate need for them.

The foundation of our scheme is the ability of the kernel (or of the user-level RNIC, through the kernel) to request that user-level buffers be de-registered. Such **kernel-induced de-registrations** require the attention and collaboration of the application or the user-level I/O library which has an incentive to comply in order to avoid RDMA address translation errors in the future. In more detail, our scheme consists of (a) *a kernel upcall interface* that notifies user-level I/O libraries of imminent NIC TLB entry replacement actions; note however, that the actual replacement is performed at a future point in time; (b) *a new registration API* that enables user processes to (optionally) wait until a registration request is satisfied, if not immediately possible; and finally (c) application-specific NIC TLB replacement policies.

The mechanics of our collaborative protocol depend on the type of buffer considered. For messaging buffers, the RNIC simply removes a given buffer address translation from its TLB and notifies (through the kernel) the user-level process of this event with an upcall. After receiving the upcall, the user-level I/O library is expected to remove the buffer from its Rx or Tx rings (Figure 1) and also to perform any necessary protocol-specific actions, such as to adjust any application-level flow control protocol state [13]. Subsequently, the user-level process I/O library may either decide to remove that buffer from its active communication pool or attempt to re-register it again in the future. Incoming I/O transfers are not affected since the RNIC ensures that such transfers use other registered buffers from that process’s communication pool.

Kernel-induced de-registrations of RDMA buffers require a similar degree of collaboration but are somewhat more complex. Unlike messaging buffers, where the RNIC has control over which messaging buffer is used next to place incoming data (and can thus ensure that a buffer is taken out of active use), RDMA buffers can be targeted at any time without the RNIC being able to predict it. As a result, the RNIC cannot immediately de-register a user-

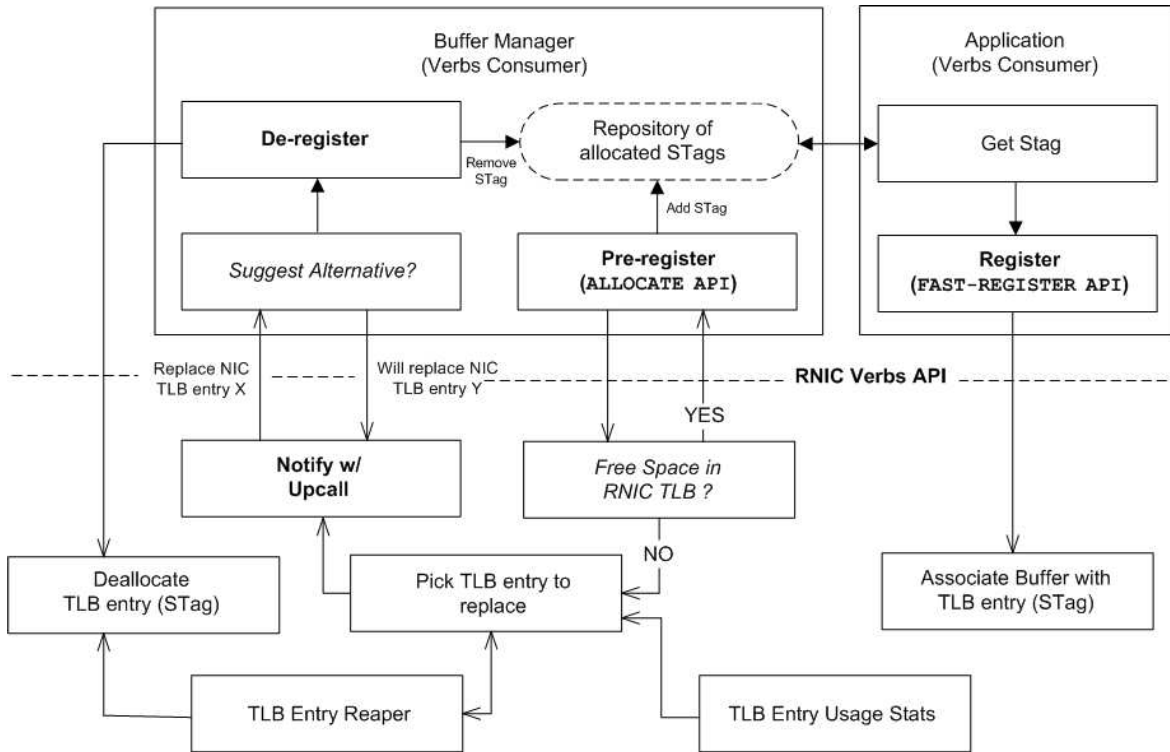

Figure 3. Collaborative upcall-based protocol. This design assumes the availability of ALLOCATE and FAST-REGISTER RNIC Verbs APIs [18] which separate the allocation of TLB resources from their binding to a particular buffer.

level buffer associated with a given RDMA or otherwise risk causing the failure of the RDMA operation. To deal with this problem, the RNIC notifies the user-level process (through the kernel) with an upcall of its intention to de-register a buffer, but provides a *grace period* T to ensure that the user-level library has enough time to de-register and decommission the buffer. T is thus the time between notification to the user-level process that a translation is about to be removed from the NIC TLB and the time of its actual removal. In this way, the kernel minimizes (but does not eliminate) the risk of failing an RDMA. The user-level I/O library is aware of the length of the grace period T and must ensure that the expected duration of any individual RDMA transfer operation it issues does not exceed it.

Given the delay in satisfying eviction requests by processes due to the grace period T and wanting to ensure that processes do not wait too long before their registrations requests are honored, the RNIC must be proactive in evicting RNIC TLB entries (note similarity to policies for cleaning dirty buffers in file caches [15]). This is the task of the “NIC TLB Reaper” module in Figure 3. The upcall notification provides only a suggestion to the application as to which TLB entries to replace; the latter may respond by offering another “victim” buffer as will be discussed later.

New Registration API: The registration interface offered by current commercial NICs returns successfully only if RNIC TLB resources are immediately available for allocation. If there are no available TLB entries at the time of the call (or if the process’s static limit is reached), the registration API returns immediately with an error code. Such an error should be treated as fatal for the application, under the assumption that there is no dynamic mechanism to free up RNIC TLB space other than processes voluntarily de-registering some or all of their buffers.

A more appropriate API for the functionality described in this paper is one where a process is given the option to wait (synchronously or asynchronously) for free RNIC TLB entries. Such an API could be termed REGISTER-WAIT. Since a registration request can trigger replacement of other TLB entries (as seen in Figure 3), a registration request might incur a delay but will be eventually honored. The delay in satisfying a registration request depends on the latency in satisfying eviction notification upcalls delivered to other processes by the kernel. Such an API has also the additional advantage of giving the kernel an estimate of the current demand of a process for RNIC TLB entries.

The “Buffer Manager” module in Figure 3 exhibits the structure of a user-level buffer manager that can handle

de-registration upcalls from the RNIC (through the kernel), and perform (pre-)registrations (*i.e.*, allocations of RNIC TLB entries). This diagram assumes the availability of the RNIC Verbs API [18].

RNIC TLB Replacement Policies: The RNIC TLB replacement policies must ensure high TLB utilization when used by multiple processes. TLB replacement must be performed proactively to reduce the average wait time incurred by processes wanting to register their buffers. A replacement policy must balance the desire to maintain a certain level of free TLB space with the amount of "pain" caused to communicating processes by asking them to de-register some of their TLB entries. By reducing the number of buffers on which a process can perform RDMA transfers, the RNIC is effectively limiting the total bandwidth achievable by that process.

In accordance with well-tested systems principles, a NIC TLB replacement policy must satisfy the following requirements:

1. Evict the least recently (or frequently, etc.) used TLB entry. To make this decision, the kernel needs information about access statistics for RNIC TLB entries, provided by the RNIC (*e.g.*, the "TLB Entry Usage Stats" module in Figure 3).

2. Consider evicting an entry that the user-level I/O library or the application considers to be less important than the one chosen by the kernel. This is reminiscent of extensible page replacement, for example as used in VINO ([21], Section 4.2) and other systems. This task is expressed by the "Consider Alternative?" module in Figure 3.

Note that even when the RNIC does not provide access statistics about its TLB entries, processes themselves (possibly via shared memory segments between user-level communication libraries and the kernel) can provide that information to the kernel to enable an LRU or LFU or other replacement algorithm. It is in the best interest of processes to be truthful regarding their temporal and spatial access characteristics as that will result in correctly applying the kernel NIC TLB replacement policies for better performance. In the absence of any such information, a simple (*e.g.*, random) replacement policy can be used.

The scheme described here works fine if the process is always aware of when an RDMA may take place, either by initiating it, or by explicitly requesting its occurrence through RPC request-response messages. However, this excludes the case where RDMA can be initiated by a remote party at any time; in such case, evicting an RDMA address translation from the local RNIC TLB may cause an I/O failure when a remote party decides to initiate RDMA using that address. One solution to this problem is to contact all remote hosts that might have and use this translation and request to invalidate their relevant RDMA reference. Another solution (that requires advanced RNIC functionality) is to allow such faults to happen and recover from them by throwing and catching remote RDMA exceptions [12]. In most systems today, the process that exports remotely-accessible communication buffers is either the initiator of RDMA or explicitly requests that remotely-initiated RDMA take place, making the mechanisms described in this paper widely applicable.

Dealing with Failure: The grace period T allowed by the kernel before a process de-registers a buffer is expected to be sufficient to avoid failure by compliant user-level I/O libraries under the assumption that a pending RDMA operation on the buffer can complete within time T after receiving the notification upcall. However, RDMA operations may suffer indeterminate delay in the network and communication libraries may not have the real-time features that enable timely processing of RDMA completions. While the latter problem can be tackled programmatically with appropriate hardware, OS, or runtime support, the RDMA network delay may be hard to bound, particularly in large-scale networks such as the Internet. This can result in evicting a busy RNIC TLB entry, leading to failure of the RDMA operation. In that case, the user-level library must be able to handle and gracefully recover from such failure. The likelihood of this happening can be reduced by increasing the grace period T at the expense of increasing the maximum delay incurred in de-registering a RNIC TLB entry. This tradeoff seems to be an unavoidable price to pay in our collaborative memory management protocol for user-level RNICs.

4. Discussion

Comparing the two options considered in this paper, it becomes apparent that one of them (using RDMA only in kernel-resident I/O subsystems) has a number of advantages over the other. First, de-registration of NIC TLB entries with kernel-based RDMA is fully controlled by the kernel; this eliminates the (however small) possibility of RDMA failures, which cannot be excluded with the upcall-based user-level RNIC approach. Second, with kernel-based RDMA, part of the complexity of managing RDMA memory resources is encapsulated in the kernel, simplifying application I/O libraries. Understandably, a source of complexity of the upcall-based scheme stems from the difficulty that programmers have with rationally arguing about time. It is fair to say, however, that this difficulty has not prevented other systems to successfully introduce time-based mechanisms

(*e.g.*, see distributed consistency based on *leases* [9]). Part of the complexity in our scheme can be encapsulated in buffer managers developed to the new registration API. Taking the above into account and given that there is no performance-related reason against a kernel-based interface to an RDMA NIC for network-storage-intensive workloads [14], we recommend the kernel-based RNIC design for practical implementations.

Note that the same general approach followed in the collaborative protocol of Section 3 can be used in other problems of similar flavor. One example is the case of managing a shared connection pool to a database or other transactional service. In that case, access to a limited number of connections by a large number of client applications must be arbitrated in a similar manner. Similar to the problem addressed in this paper, lightly-used connections may be taken away from certain clients by the connection manager and given to other clients who request them. To reduce the possibility of failure (*i.e.*, forcing a connection closed before a client process is finished using it) a similar grace period must be granted before a connection is actually taken away.

5. Related work

A number of recent RDMA protocol specifications, such as InfiniBand [10], the iWARP protocol suite [19], and RNIC [18] are addressing the issue of memory management and particularly how to reduce the cost of memory registration so that the association of NIC TLB resources with a buffer's memory address translation can be done on-the-fly prior to each RDMA I/O. This is a key requirement of storage protocols such as iSCSI and its extensions for RDMA [6] (iSER), where the binding between a memory buffer and the physical memory backing it is not known until the time of I/O. The FAST-REGISTER MEMORY REGION and BIND MEMORY WINDOW APIs developed for this purpose, split the process of registration in two parts; the allocation of NIC resources, such as TLB entries, protection checks, etc.; and the loading of an address translation into those resources. The second part can be performed just prior to (and on the data-path of) the I/O. This requires that NIC TLB allocations be made early, possibly at the time of connection initialization, and memory region identifiers (`STags`) allocated in advance. The two above interfaces are provided at the *Verbs* [18] level and are meant to be implemented at the hardware, NIC firmware, driver or user-level network library. The current practice for using the FAST-REGISTER MEMORY REGION API is to rely on a pool of pre-allocated memory regions, which is currently assumed to be fixed. To work with the scheme described in this paper, the pool of pre-allocated regions must be managed dynamically by a buffer manager as depicted in Figure 3, which must ensure that new registrations (in this case, allocations of RNIC TLB space) are carried out as needed to compensate for any kernel-induced de-registrations. It is assumed that prior to issuing a FAST-REGISTER MEMORY REGION work request followed by an RDMA work request, the application must ensure that the `STag` used is still registered with the NIC, by validating it with the buffer manager.

Previous work on memory management for RNICs and on reducing RNIC TLB memory requirements has focused on extending RNIC address translation structures into host memory, maintaining only a cache of address translations (and potentially of other state, such as network connections) onboard the RNIC. The key benefit of such *two-level memory management* schemes is improved scalability in the amount of memory that can be registered with the RNIC with the potential to support practically *memory-free* RNICs. Two-level schemes include systems such as U-Net [24], UTLB [7], and miNI [1]. In addition, the work of Schoinas and Hill [20] explored the design space of two-level memory management schemes for NICs. Design issues that vary in these projects are (a) whether the host-side (*i.e.*, 2nd level) address translation structure is accessed by the RNIC or by the host; (b) whether access to host memory is by DMA or by programmable I/O (in case of direct access by the NIC), or by interrupts thrown by the NIC and handled by host software; and other issues such as time of memory pinning, etc. There are however, a number of factors that have impeded the adoption of two-level schemes: First, there is significant complexity involved in maintaining such a two-level address translation structure. Second, given the performance penalty of accessing host memory over I/O buses, it seems likely that effective use of the RNIC TLB will still be the key factor affecting performance, which argues for increasing RNIC memory resources. The *kernel-induced de-registration scheme* described in Section 3 is a similar but simpler and more practical alternative to the above referenced two-level memory management schemes.

6. References

[1] R. Azimi, A. Bilas, "miNi: Reducing Network Interface Memory Requirements with Dynamic Handle Lookup", in *Proceedings of the 17th ACM International Conference on Supercomputing (ICS03)*, June 2003.

[2] M. Blumrich and K. Li and R. Alpert and C. Dubnicki and E for the SHRIMP Multi-computer", in *Proc. of 21st Annual International Symposium on Computer Architecture (ISCA)*, Chicago, IL, April 1994.

[3] P. Buonadonna, D. Culler, "Queue-Pair IP: A Hybrid Architecture for System Area Networks", in *Proc. of 29th Annual International Symposium on Computer Architecture (ISCA)*, Anchorage, AK, May 2002.

[4] G. Buzzard, D. Jacobson, S. Marovich, J. Wilkes, "An Implementation of the Hamlyn Sender-Managed Interface Architecture", in *Proc. of Second Symposium on Operating Systems Design and Implementation (OSDI)*, Seattle, WA, October 1996.

[5] B. Callaghan, T. Talpey, "NFS Direct Data Placement", IETF draft-ietf-nfsv4-nfsdirect-00.txt, July 2004.

[6] M. Chadalapaka, U. Elzur, M. Ko, H. Shah, P. Thaler, "A Study of iSCSI Extensions for RDMA (iSER)", in *Proceedings of ACM NICELI 2005 Workshop*, Karlsruhe, Germany, August 2003.

[7] Y. Chen, A. Bilas, S. Damianakis, C. Dubnicki, K. Li, "UTLB: A Mechanism for Address Translation on Network Interfaces", in Proceedings of 8th ASPLOS, San Jose, CA, October 1998.

[8] P. Druschel, V. Pai, W. Zwaenepoel, "Extensible Kernels are Leading OS Research Astray", in *Proceedings of Workshop on Hot Topics in Operating Systems (HotOS)*, 1999.

[9] J. Gray, D. Cheriton, "Leases: An Efficient, Fault-tolerant Mechanism for Distributed File Cache Consistency", in Proceedings of ACM Symposium on Operating Systems Principles, Litchfield Park, AZ, December 1989.

[10] InfiniBand Trade Association, http://infinibandta.org/specs

[11] F. Kaashoek, McKenzie, D. Engler, G. Ganger, H. Briceno, Hunt, D. Mazieres, T. Pickney, R. Grimm, J. Giannotti, "Application Performance and Extensibility in Exokernel", in *Proceedings of 16th ACM Symposium on Operating Systems Principles*, Saint Malo, France, 1997.

[12] K. Magoutis, S. Addetia, A. Fedorova, M. Seltzer, "Making the Most of Direct Access Network Attached Storage", in *Proceedings of 3rd USENIX Conference on File and Storage Technologies*, San Francisco, CA, April 2003.

[13] K. Magoutis, S. Addetia, A. Fedorova, M. I. Seltzer, J. S. Chase, A. Gallatin, R. Kisley, R. Wickremesinghe, E. Gabber, "Structure and Performance of the Direct Access File System", in *Proceedings of the USENIX Annual Technical Conference*, Monterey, CA, June 2002.

[14] K. Magoutis, M. I. Seltzer, E. Gabber, "The Case Against User-Level Networking", in *Proceedings of 3rd Workshop on Novel Uses of System-Area Networks (SAN-3)*, Madrid, Spain, February 2004.

[15] M. K. McKusick, K. Bostic, M. Karels, J. Quarterman, "The Design and Implementation of the 4.4BSD Operating System", Addison Wesley, 1996.

[16] J. Mogul, "TCP Offload is a Dumb Idea Whose Time Has Come", in *Proc. of Ninth Workshop on Hot Topics in Operating Systems (HotOS-IX)*, Lihue, Hawaii", May 2003.

[17] M-VIA: A High Performance Modular VIA for Linux, National Energy Research Scientific Computing Center, http://www.nersc.gov/research/FTG/via, 1999.

[18] R. Recio, "RDMA enabled NIC (RNIC) Verbs Overview", RDMA Consortium, April 29, 2003; http://www.rdmaconsortium.org/home/RNIC_Verbs_Overview2.pdf

[19] R. Recio, "An RDMA Protocol Specification", Internet Draft, draft-ietf-iwarp-rdma-01.txt, February 2003.

[20] I. Schoinas, M. Hill, "Address Translation Mechanisms for Network Interfaces", in *Proc. of Fourth International Symposium on High-Performance Computer Architecture (HPCA)*, February 1998.

[21] M. Seltzer, Y. Endo, C. Small, K. Smith, "Dealing with Disaster: Surviving Misbehaved Kernel Components", in *Proceedings of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 1996.

[22] Virtual Interface (VI) Architecture Specification, December 1997.

[23] T. von Eicken, A. Basu, V. Buch, W. Vogels, "U-Net: A User-Level Network Interface for Parallel and Distributed Computing", in *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, Copper Mountain, CO, December 1995.

[24] M. Welsh, A. Basu, T. von Eicken, "Incorporating Memory Management into User-level Interfaces", in Proceedings of Hot Interconnects V, Stanford, CA, August 1997.