Design and Implementation of the iWarp Protocol in Software

Dennis Dalessandro
Ohio Supercomputer Center
1 South Limestone St., Suite 310
Springfield, OH 45502
dennis@osc.edu

Ananth Devulapalli
Ohio Supercomputer Center
1 South Limestone St., Suite 310
Springfield, OH 45502
ananth@osc.edu

Pete Wyckoff
Ohio Supercomputer Center
1224 Kinnear Road
Columbus, OH 43212
pw@osc.edu

Abstract

The term iWarp indicates a set of published protocol specifications that provide remote read and write access to user applications, without operating system intervention or intermediate data copies. The iWarp protocol provides for higher bandwidth and lower latency transfers over existing, widely deployed TCP/IP networks. While hardware implementations of iWarp are starting to emerge, there is a need for software implementations to enable offload on servers as a transition mechanism, for protocol testing, and for future protocol research.

The work presented here allows a server with an iWarp network card to utilize it fully by implementing the iWarp protocol in software on the non-accelerated clients. While throughput does not improve, the true benefit of reduced load on the server machine is realized. Experiments show that sender system load is reduced from 35% to 5% and receiver load is reduced from 90% to under 5%. These gains allow a server to scale to handle many more simultaneous client connections.

1 Introduction

The race between increasing network speeds and increasing processor capability has been on for decades. Current generation commodity processors are capable of keeping a gigabit network pipe full, but fall short when presented with a ten gigabit network. Traditional packet processing requires many calculations and memory operations by the CPU in a computer hosting a network interface card. This processing limits the achievable data throughput of faster network technologies. Furthermore, network processing demands compete with user applications for available CPU cycles. It is possible, especially in ten gigabit networks, that the CPU spends more time on network processing than on real computations, in effect starving the application.

To avoid this CPU bottleneck, mechanisms such as Remote Direct Memory Access (RDMA) [16] have been developed. RDMA allows network adapters to move data directly from one machine to another without involving either host processor. A number of research projects have used RDMA to increase network performance, including U-Net [21], Hamlyn [2], and EMP [20]. Commercial products are available as well, such as VIA [7], Infiniband [9], Quadrics [15], and Myrinet [13].

A second mechanism that avoids CPU overheads is Operating System Bypass. This is a scheme in which the user application interacts directly with the network card, avoiding overheads from system calls, hardware interrupts, and context switches. As a result, latency and overhead of data motion is significantly reduced.

iWarp is a recent protocol specification that serves as a convergence point for these techniques and previous research efforts. It is an RDMA implementation on top of existing Internet protocols, namely TCP or SCTP on IP, giving it the major advantage of being compatible with the existing Internet infrastructure. iWarp uses both RDMA and OS bypass to move data without the CPU or operating system being involved, greatly increasing performance.

The other desirable feature of iWarp is protocol offload. The RDMA-capable network adapter also handles all network processing, similar to a TCP Offload Engine (TOE). As the network adapter deals with most aspects of network processing, the CPU can continue application processing concurrently with network transfers.

Specifications for the iWarp protocol and software stack exist as IETF RFCs, and 1 Gb/s and 10 Gb/s hardware products are available at present.

Commodity implementations of gigabit iWarp hardware, such as the Ammasso 1100 [1] are beginning to appear, as well as a 10 gigabit implementation from Chelsio [3]. At present the cost per port for a 10 gigabit Ethernet switch is very high. In fact, more expensive than an InfiniBand switch. While 10 gigabit may be too costly for most installations, gigabit switches are relatively cheap, and common place. Given the adoption history of gigabit Ethernet [22] it is clear that the cost of 10 gigabit Ethernet will drop.

In the next section we discuss the motivation for a software iWarp stack, followed by an in-depth description of the iWarp specification. The rest of the paper is dedicated to describing our software design and implementation, as well as measuring its performance.

Support for this project was provided by the Department of Energy ASC program and Sandia National Laboratories.

2 Motivation

There are many reasons to develop a software implementation rather than using an existing or new hardware implementation. The first motivating factor is to enable offload processing on servers. Since iWarp-capable Ethernet adapters are only beginning to enter the market, at somewhat high prices compared to non-iWarp devices, it is likely that organizations will choose to put iWarp adapters in only the most heavily loaded servers. This is where the offload features will provide the most benefit. However, both sides of a connection must agree to use the iWarp protocol, otherwise the data transmission must use the traditional socket semantics, requiring multiple data copies as well as more CPU cycles.

Although a software implementation of iWarp on a client machine should not be expected to require less processor utilization or to provide better performance, it will enable the server with which the client communicates, to realize the benefits of having a hardware iWarp adapter.

The second expected use for this work is as a vehicle for future research into the iWarp protocol and its application. Due to the fact that few iWarp hardware components are available on the market, little in-depth work has been done with them for particular user communities, such as high-performance computing, file and block storage, and transaction processing. The impacts on applications, higher-layer protocols, system-area networks, wide-area networks and many other aspects have yet to be analyzed. In having a user-space implementation of iWarp, we can deploy the software on arbitrary machines at no cost, and study such aspects at a large scale. This advance planning is crucial, both to influence evolving hardware designs and to plan for future adoption of high-speed protocol-offload NICs in our computing environments.

Finally, having an open-source implementation of the iWarp protocol that is fully compliant with existing specifications will permit testing of a wide array of applications and scenarios, as well as give developers a head start on moving applications to iWarp devices. The code can be used to verify conformance of new hardware implementations as it is easily customizable to generate particular, perhaps erroneous, responses in a given situation to analyze behavior of the device under test.

3 iWarp

The terms RDMA over TCP/IP, RDMA over Ethernet, and iWarp all refer to the same thing: a zero-copy, OS-bypass mechanism for implementing Remote Direct Memory Access (RDMA) operations over a standard TCP/IP-based network.

To facilitate RDMA, special hardware is required in the form of an RDMA-enabled Network Interface Card (RNIC). The cables and switching infrastructure are generic

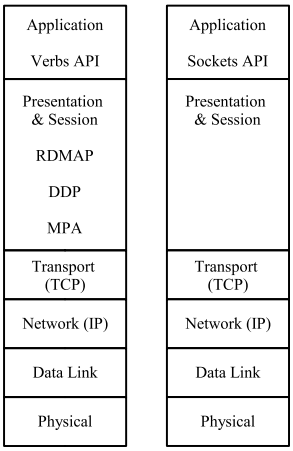

Figure 1. iWarp and TCP Protocol Stacks.

Ethernet equipment. In making use of the existing network infrastructure, iWarp has a unique capability that its special purpose counterparts can never match, that is the ability to work in the wide area network.

The iWarp protocol stack is shown in Figure 1. It consists of a verbs or API layer [8], an RDMA protocol layer (RDMAP) [17], a direct data placement layer (DDP) [19], and a Marker PDU Aligned (MPA) Framing layer [4]. MPA is built on top of standard TCP/IP layers.

The motivation for the RDMAP, DDP and MPA layers in the stack is to facilitate reliable communication with hardware that offloads processing of the TCP/IP stack, such as an RNIC. The intention may be for this processing to be done in hardware, but the specification does not restrict us to hardware only, and as such software implementations are also potentially of value.

The verbs or API layer is the layer that is exposed to user applications. The key advantage of the verbs layer is to provide direct access to the RNIC hardware. Native RDMA operations are available via the verbs to any application. iWarp has the advantage of a verbs specification. This means that while there is no specific API that a hardware vendor must provide, there is a specification for creating APIs that explicitly states which functions must be available, the parameters, and the semantics of the operation. Having a verbs specification is also a hindrance. It means every vendor potentially has a different API. Luckily we have alternatives in the form of middleware APIs such as the Message Passing Interface (MPI) [12], which is the *de facto* form of message passing in the HPC world, as well as the Direct Access Programming Library (DAPL) [6]. DAPL is an API that enables code written for one particular device to work on another that is DAPL compliant. For instance, code written for a particular iWarp implementation can be run on an InfiniBand implementation, and vice versa, requires very minimal changes, in most cases just one line of code.

The verbs layer communicates with the RDMAP layer which provides the semantics for the RDMA operations. RDMAP [17] provides read and write services directly to applications and enables data to be transferred directly into application buffers without intermediate data copies. It also enables an OS-bypass implementation.

The Direct Data Placement Protocol (DDP) [19] enables an upper-layer protocol, in this case RDMAP, to send data to a host without requiring the receiver to place the data initially in an intermediate buffer. This can enable the receiver to consume substantially less memory bandwidth than a buffered model because it is not required to move the data from the intermediate buffer to the final destination. This model also saves CPU cycles that would otherwise be required to copy the data and removes the CPU bandwidth limitation on network performance.

The DDP protocol is a message-oriented protocol, while TCP is a streaming byte-oriented protocol, thus a framing mechanism is necessary to detect record boundaries. The MPA specification [4] describes one such framing mechanism that uses periodic markers in the byte stream to facilitate record detection. It also specifies a stronger CRC-32c data integrity calculation [18] due to the inadequacy of the TCP CRC.

4 Design

In this section we give an overview of some of the design issues that we faced during the implementation of our iWarp software stack. One of the most fundamental design choices was the API that we would export to user applications. We have chosen to base our API off of the original verbs specification [8] from the RDMA Consortium [16]. We decided not to go directly with a DAPL [6] or an SDP [14] API due to complexity, also it has been shown that [5] running sockets over a fast RDMA based network does not necessarily allow for the full performance of the network.

In implementing a subset of the most necessary verbs we are in turn able to limit ourselves to a subset of RDMAP layer communication primitives. The DDP and MPA layers are functionally complete and are able to produce TCP packet aligned MPA frames for transmission, along with doing the CRC-32c calculation and validation.

Another important design issue was the amount of interdependency between the verbs API layer and the underlying iWarp software stack. A carefully planned interface between the verbs and the underlying software stack has been developed with a minimal number of functions available to the verbs. This facilitates changes in the underlying code base. It is thus important that the only layer the verbs may interface with is RDMAP.

Naturally, the data structures shared between the verbs and the software stack is a key design decision as well. Thus the only data structure shared by the verbs and the lower layer (RDMAP) is the Completion Queue (CQ). The lower layer will produce an entry onto the CQ, while the verbs layer then consumes the entry.

5 Implementation

Our implementation has two main modules, one for the iWarp stack and another for the verbs layer. As discussed in previous sections, the iWarp stack can be either in the kernel or userspace, but the verbs must be in userspace. We have currently implemented both verbs and iWarp stack in user space for use with typical non-kernel-resident applications.

An ideal implementation would have a receive thread and a send thread for maintaining independent progress for both sends and receives. But since multi-threading would increase complexity and the validation time, for the initial implementation we restricted ourselves to the single-threaded model. As a result, to ensure receive progress, the receiver must call a progress function, so that messages continue to arrive. The verbs layer exports the progress function to the user application for this purpose.

Sends are implemented as blocking but receives are implemented as non-blocking. For sends, we tried to maintain a close resemblance to non-blocking semantics of posting a work request and polling for the request. Sends and receives are mapped directly to underlying TCP function calls exported by the kernel. The choice of single threaded model constrained us to blocking sends.

The MPA specification [4] lays out the negotiation protocol for binding DDP with MPA. During our implementation we found that one of our hardware iWarp adapters, the Ammasso 1100 [1], does not yet support MPA markers. As a result we had to construct a generalized implementation of the MPA connection setup phase. Our implementation tolerates such limitations of hardware, by following the least common capabilities that hardware can handle. We also permit disabling the CRC calculation to isolate the effect it has on performance.

On a DDP send, as initiated by an send, RDMA write, or RDMA read request from the upper layer, our DDP library generates multiple DDP segments. Each segment contains a DDP header that describes the disposition of the message to the receiver. Each segment is smaller than the payload size of the underlying reliable transport. Our implementation uses the kernel system calls `socket()`, `bind()`, `listen()` and `connect()` through the C library to set up a TCP connection.

Since iWarp is a zero-copy protocol, we want to limit any copies of data introduced by our software only version. We ensure that no additional copies are made in our software iWarp stack. This makes the stack more efficient and conforms more to the specification. Since our initial implementation is in user-space there is a copy involved to kernel space, both at sending and receiving ends. This is unavoid-

able when using the kernel sockets interface to TCP.

In a hardware implementation, the RNIC would handle incoming DDP segments, placing them automatically in user memory, and signal the application when the entire message is complete. The delivery mechanism is unspecified, and could be a processor interrupt or adding an entry to a completion queue somewhere. (The RDMA layer defines completion, but leaves the mechanism up to the implementation, hinting that a queue is a likely way to do it.) Our software implementation uses a `poll()` system call to wait until data arrives on the socket from the TCP/IP stack and network card. It then reads the DDP header into a small buffer, inspect the packet, determine the data destination, and read the rest of the segment directly into the appropriate memory location in the application.

6 Experiments

In this section we describe the experiments that we have conducted in order to measure the performance of our software iWarp stack. We perform three sets of tests: latency, bandwidth and CPU utilization. Latency and bandwidth are the basic metrics by which network performance is analyzed. Considering CPU utilization shows the effects of iWarp protocol features on reducing host processor load. We show that it is possible for a host running a hardware iWarp implementation to benefit from the zero-copy protocol offload provided by its hardware when communicating with a host running a software iWarp implementation.

Our experimental testbed consists of two IBM x-series servers, each equipped with a single 2.8 GHz Pentium 4 Xeon, with 1.5 gigabytes of RAM. The on-board Intel 82546EB copper gigabit ethernet controller was utilized for all software tests. Each server is also equipped with an Ammasso 1100 RNIC [1]. The Ammasso cards are early FPGA-based versions on loan for evaluation from Ammasso Inc.

Various configurations of network cards were used, both the Ammasso RNICs (denoted "hw" in the plots) and traditional ethernet with our iWarp software implementation ("sw"). TCP/IP-only tests were performed as well. All of these tests were conducted with the devices connected back-to-back via a standard Cat5 Ethernet crossover cable.

6.1 Latency Test

The latency referred to is the half round trip (RTT/2) time it takes for a ping-pong of a particular size. The ping is made via an RDMA write as is the pong. Each side polls the last byte of its buffer to determine when an RDMA write has completed. This ping-pong process is repeated a number of times, 50 iterations in our case, and the average and standard deviation are computed. These values are reported in Table 1.

The results reflect a few interesting points. Given the latency for TCP to TCP as a baseline, we see that our iWarp software stack adds negligible additional latency for small messages. For large messages, there is a small overhead to the iWarp layer when CRC is off, and a considerable expense (about 50% longer) to perform a CRC calculation on either the sender or receiver. Hardware implementations are not affected since the CRC is performed in hardware as the data is received or transmitted, as evidenced in the hardware-only results table.

	4 byte messages	64 kB messages
hw→hw	15.0 ± 0.9	609.7 ± 5.9
sw→sw (crc off)	62.7 ± 7.2	687.5 ± 27.6
sw→sw (crc on)	62.2 ± 1.3	1401.9 ± 265.5
hw→sw	62.8 ± 2.9	950.4 ± 45.9
sw→hw	61.2 ± 2.8	937.9 ± 16.0
tcp→tcp	62.7 ± 4.7	624.5 ± 23.0

Table 1. Half-round trip latencies (μs)

The CRC calculation also has an impact on the hybrid tests, either software iWarp sending to a hardware receiver or *vice versa*. Due to the fact that the Ammasso RNIC does not negotiate per spec we are forced to use a CRC calculation on the software side as well. Again we see that the latencies for small messages are not affected much by the CRC calculation, but the larger message shows an impact.

6.2 Throughput Test

To examine the throughput of our implementation, we use a unidirectional spray test where the sender initiates RDMA writes to the receiver as fast as it can. The receiver polls on the last byte of the receive buffer, waiting for the value to change that indicates that the last RDMA write has completed, then sends a four-byte message to the sender indicating completion. The sender measures the overall time to complete the RDMA writes and receive this small completion message and calculate the throughput (in millions of bits per second). The results are shown in Figure 2.

The top curve in the figure is raw TCP. It represents the lowest point in the software stack and achieves the highest possible throughput. Just below that is the configuration where the hardware RNIC serves as both the sender and receiver. It performs the CRC calculation and verification as required by the specifications as mentioned earlier. Adjacent to this curve is the software-to-software configuration, with the CRC disabled, showing that our implementation is able to achieve the full line rate of the gigabit Ethernet network. When our software implementation is set to enable CRC, performance drops about 100 Mb/s. Identical results are achieved when the hardware RNIC is the sender and our software implementation receives packets; however, the reverse case performs better, demonstrating that the CRC validation on receive is more expensive than the generation at transmit time.

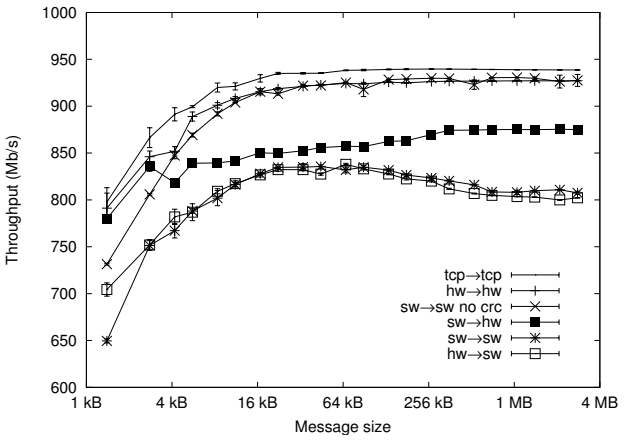

Figure 2. Throughput versus message size.

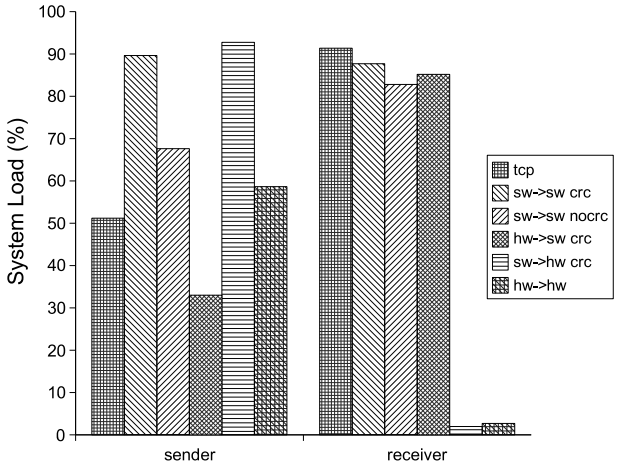

Figure 4. Small message system load (4 kB).

6.3 System Load

The system load, which accounts for both user and kernel CPU time and memory subsystem load, was measured for the throughput experiments described above. A low-priority background task is spawned which runs concurrently with the process whose load is to be measured. The load is computed subtractively by measuring the performance degradation of the background task relative to when the background task is run by itself. Figures 3 and 4 show the plots of relative system load at sender and receiver for large and small messages respectively.

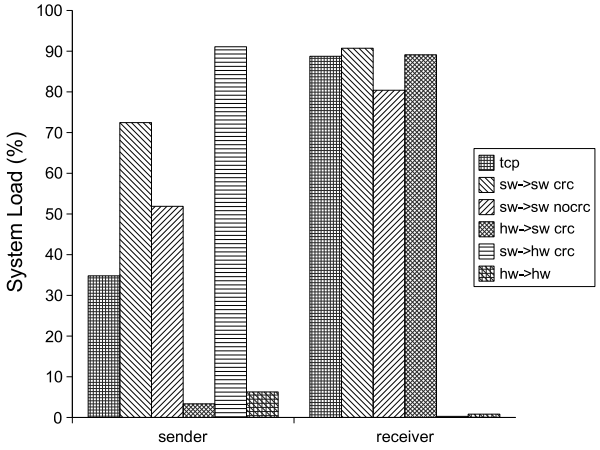

Figure 3. Large message system load (68 kB).

The benefits of hardware offload are clearly identified in the very low utilization on the sender side when the RNIC is the sender. The corresponding TCP sender load is larger, around 35%, and adding our software iWarp stack brings that up to 50%. Calculating the CRC raises the total to 70%.

For receives, the load required by the hardware receiver is almost unmeasurable, even in the case when a software client is sending the messages. Both the TCP and software iWarp receivers load the system almost completely as they copy messages from the network through the kernel to their destination application buffers.

Compared to large messages, the load at the sender is a bit higher in the software case due to the increased load from more frequent packet processing for a given transmission rate, around 10–20% in our results. The hardware case also shows an increase due to more work required of the CPU to initiate message transmission.

For small messages, software iWarp with CRC enabled is about 40% more expensive than TCP at the sender. Without CRC, the system load is about 20% more expensive than TCP. In the case of hardware sender to software receiver, the system load appears to be less than the pure hardware case, but this configuration achieves less throughput leading to less work per unit time for the sender.

On the receiving side, all software iWarp and TCP cases show nearly 90% system load, because the common underlying TCP packet and header processing on receive are CPU intensive. Differences due to CRC calculation are not seen due to the domination of TCP packet processing. When the hardware implementation servers as receiver, its load is near zero regardless of sender.

These results demonstrate that a hybrid implementation of software and hardware iWarp provides a low-cost means of implementing iWarp while still exploiting the advantages of network offload.

7 Related Work

There has been extensive research into zero-copy, OS-bypass, and RDMA networking [21, 2, 20]. We take advantage of this work to build a software implementation of the iWarp suite of specifications that permits an iWarp-enabled server to take full advantage of its network interface card.

Other technologies [7, 9, 15, 13] use similar techniques to accelerate networking, but, being specialty hardware can not take advantage of the existing installed Ethernet base.

Much work into appropriate application programming interfaces (APIs) is underway. The DAT Collaborative's DAPL [6] API, and the InfiniBand verbs [9] are two well known examples, and have had an influence on our choice of function signatures. The IT-API [10] is similar to the InfiniBand verbs but attempts to accommodate a wider range of technologies, including both iWarp and IB.

King and Berry [11] analyze system overheads incurred in processing of a software iWarp stack to determine the feasibility of deploying non-offload solutions. They project performance to near-future commodity CPUs and 10 Gb/s networks and show that the CRC calculations consume the overwhelming majority of processor cycles. Their experimental environment was a processor simulator to allow gathering detailed information while our work is aimed to real-world deployment.

8 Future Work

To enable use of the iWarp software implementation by in-kernel clients, we will adapt the user-space version to produce a kernel-space implementation. This will permit exploration of the use of iWarp for many applications such as NFS for file storage and iSER (iSCSI over RDMA) for block storage. This will reduce the system call overhead. For maintaining independent progress of send and receives, multi-threading is a natural choice. Multi-threading will also enable non-blocking sends and receives. We are currently in the process of migrating to a multi-threaded implementation of iWarp stack.

9 Conclusion

The work presented here allows a server with an iWarp network card to take full advantage of the zero-copy and protocol offload provided by the card even when clients are not equipped with such a device. While throughput will necessarily be lower if the clients are using a software implementation, the true benefit of reduced load on the server machine can still be realized. Experiments demonstrated that for large message sizes, load for transmission is reduced from 35% to 5% and for reception is reduced from 90% to under 5%. These gains allow a server to handle many more client communications. Other uses for this work are in protocol validation and upper-layer protocol and application research.

References

[1] Ammasso Inc. Ammasso 1100. URL `http://www.ammasso.com/products.htm`.

[2] G. Buzzard, D. Jacobson, M. Mackey, S. Marovich, and J. Wilkes. An implementation of the Hamlyn send-managed interface architecture. In *Proc. of OSDI'96*, October 1996.

[3] Chelsio Communications. T210 10GbE protocol engine. URL `http://www.chelsio.com/products/T210.htm`.

[4] P. Culley, U. Elzur, R. Recio, S. Bailey, and J. Carrier. Marker PDU aligned framing for TCP specification, February 2004. URL `http://www.ietf.org/internet-drafts/draft-ietf-rddp-mpa-02.txt`.

[5] Dennis Dalessandro. Why RDMA? URL `http://www.osc.edu/~dennis/rdma/rdma.html`.

[6] DAT Collaborative. Direct access transport. URL `http://www.datcollaborative.org`.

[7] D. Dunning *et al.* The Virtual Interface Architecture. *IEEE Micro*, 18(2):66–76, 1998.

[8] Jeff Hilland, Paul Culley, Jim Pinkerton, and Renato Recio. RDMA Protocol Verbs Specification, April 2003. URL `http://www.rdmaconsortium.org/home/draft-hilland-iwarp-verbs-v1.0-RDMAC%.pdf`.

[9] *InfiniBand Architecture Specification*. InfiniBand Trade Association, October 2004. URL `http://www.infinibandta.org/specs/`.

[10] Internet Software Consortium. Interconnect transport API, 2004. URL `http://www.theopengroup.org/`.

[11] Steven King and Frank Berry. Software RDMA over TCP/IP on a general purpose CPU. In *Proceedings of RAIT'04 in conjunction with Cluster'04*, San Diego, CA, September 2004.

[12] *MPI: A Message-Passing Interface Standard*. MPI Forum, March 1994.

[13] Myricom. Myrinet. URL `http://www.myri.com/`.

[14] James Pinkerton, Ellen Deleganes, and Michael Krause. Sockets Direct Protocol (SDP) for iWARP over TCP (v1.0), October 2003. URL `http://www.rdmaconsortium.org/home/draft-pinkerton-iwarp-sdp-v1.0.pdf`.

[15] Quadrics. URL `http://doc.quadrics.com`.

[16] RDMA Consortium. Architectural specifications for RDMA over TCP/IP. URL `http://www.rdmaconsortium.org/`.

[17] R. Recio, P. Culley, D. Garcia, J. Hilland, and B. Metzler. An RDMA protocol specification, April 2005. URL `http://www.ietf.org/internet-drafts/draft-ietf-rddp-rdmap-04.txt`.

[18] J. Satran, K. Meth, C. Sapuntzakis, M. Chadalapaka, and E. Zeidner. Internet small computer systems interface (iSCSI). RFC 3720 (Proposed Standard), April 2004. URL `http://www.ietf.org/rfc/rfc3720.txt`.

[19] Hemal Shah, James Pinkerton, Renato Recio, and Paul Culley. Direct data placement over reliable transports, February 2005. URL `http://www.ietf.org/internet-drafts/draft-ietf-rddp-ddp-04.txt`.

[20] Piyush Shivam, Pete Wyckoff, and D. K. Panda. EMP: zero-copy OS-bypass NIC-driven gigabit ethernet message passing. In *Proc. of SC '01*, Denver, CO, November 2001.

[21] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A user-level network interface for parallel and distributed computing. In *Proceedings of SOSP'95*, Copper Mountain, Colorado, December 1995.

[22] D. Zabrowski. 10 Gigabit Ethernet market opportunities. URL `http://www.s2io.com/news/events/webex/DaveZabrowski.pdf`.