iWarp Protocol Kernel Space Software Implementation

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

Zero-copy, RDMA, and protocol offload are three very important characteristics of high performance interconnects. Previous networks that made use of these techniques were built upon proprietary, and often expensive, hardware. With the introduction of iWarp, it is now possible to achieve all three over existing low-cost TCP/IP networks.

iWarp is a step in the right direction, but currently requires an expensive RNIC to enable zero-copy, RDMA, and protocol offload. While the hardware is expensive at present, given that iWarp is based on a commodity interconnect, prices will surely fall. In the meantime only the most critical of servers will likely make use of iWarp, but in order to take advantage of the RNIC both sides must be so equipped.

It is for this reason that we have implemented the iWarp protocol in software. This allows a server equipped with an RNIC to exploit its advantages even if the client does not have an RNIC. While throughput and latency do not improve by doing this, the server with the RNIC does experience a dramatic reduction in system load. This means that the server is much more scalable, and can handle many more clients than would otherwise be possible with the usual sockets/TCP/IP protocol stack.

1 Introduction

With the network performance bottleneck moving away from the NIC and to the CPU, new solutions for network processing are in need. Well known high performance interconnects such as InfiniBand [10], Myrinet [14] and Quadrics [18] have employed a technology known as Remote Direct Memory Access (RDMA) to solve this problem. Further they address the need for zero-copy and protocol offload.

Zero-copy is a mechanism for moving data without introducing a copy into and out of operating system buffers. Typical TCP/IP processing, as implemented in the Linux kernel for instance, involves multiple copies of user buffers, both during sending and receiving. Thus the kernel must generally be bypassed to avoid the expense of copying. Though there have been attempts to enable zero-copy without OS bypass, such as [11], these attempts have made certain assumptions that will not always hold true.

The importance of RDMA has been shown in a number of research projects, such as U-Net [23], Hamlyn [3], and EMP [21]. These efforts pioneered the use of advanced techniques, but did not result in widely available commercial products. iWarp is the next likely step in the evolution of these technologies towards viable products. iWarp enables RDMA, including zero-copy and protocol offload over the existing TCP/IP infrastructure. The easiest way of explaining iWarp is simply "RDMA over Ethernet." Being built on top of TCP/IP is iWarp's biggest advantage. This means that iWarp will work in the WAN, unlike the other special purpose interconnects previously mentioned.

Keeping with the commodity, and standards driven nature of TCP/IP, iWarp itself is governed by a set of IETF RFCs, specifically RDMAP [19], DDP [20], and MPA [4]. An application programming interface (API) is necessary for user space applications to make use of iWarp hardware, and in our case the kernel space iWarp software device. A verbs level API specification [9] is available and used as a guide by at least one vendor. There is also ongoing work to integrate iWarp into the OpenIB software stack [16]. Furthermore, the IETF has published a draft addressing iWarp security concerns [17].

Implementations of iWarp on 10 Gigabit Ethernet will likely be available from a number of companies in the near future. The first commercially available iWarp adapter, a Gigabit product from Ammasso [1] was available for the first time in mid 2004.

In addition to RDMA, other solutions have been proposed. Specifically TCP Offload Engines, or TOE cards. TOE cards address the problem by offloading the processing of the TCP/IP protocol stack to the NIC. This is a huge

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

step up from traditional TCP/IP processing in the kernel, but only addresses one side of the problem. In a TOE card, data is generally moved through the kernel, by the CPU as with non-offloaded Ethernet adapters. The requirement of copying data to and from user-space buffers adds significant load on the host processor that is not needed with RDMA.

In the next section we provide the motivation for a software based iWarp solution, followed by an in-depth explanation of iWarp. Then in section 4 we provide insight into the design and implementation challenges we faced during this work. We follow this with an analysis of experiments designed to showcase the performance capabilities of our iWarp protocol software stack, and that it is indeed possible for a hardware equipped host to experience dramatically lowered system usage when communicating with a software only host. We finish with a bit about related and future work, before drawing our final conclusions.

2 Why Software iWarp?

There are a number of reasons for implementing the iWarp protocol stack in software. The key reason comes from the observation that if a server is equipped with an RDMA Enabled NIC, or RNIC, it can only take advantage of it if the remote host also can speak the iWarp protocol. Traditionally this meant that both sides must have the hardware RNIC. With our software implementation, we create a *software* RNIC device with the same programming interface as a hardware device. The remote host does not observe a difference and as such is able to take advantage of the protocol offload and zero-copy operations provided by its own RNIC.

This approach shows no performance benefit on the host that must implement the iWarp protocol in software, in fact it even adds some additional overhead. It is important to realize that it is not our goal to provide any direct benefit to the host with software iWarp. The key advantage from our work comes from the dramatically reduced load experienced on a hardware iWarp equipped server when the client is not hardware equipped. If the server is able to take advantage of its RNIC, it is able to handle many more clients with the same processing power, or equivalently, service the same client load but also have spare cycles for other work. Viewed as an evolution from traditional Gigabit Ethernet adapter cards, to offloaded 10 Gigabit iWarp adapters, this approach has a lower deployment cost and focuses spending where it will provide measurable advantages. Only the most critical of servers need new adapters, and clients can use our freely available software.

Other reasons for implementing iWarp in software are for research into the inner workings of the protocols, and how to best take advantage of them. Also an open source implementation of the iWarp protocol allows for testing a wide variety of applications to verify compliance, where performance is not necessarily a concern but correctness on a large number of hosts is. It is possible with a software-only solution to develop applications in an infrastructure that does not have the high cost overhead of purchasing RNICs.

This work builds on our previous software iWarp implementation [5], focusing on research and experimentation beyond what was previously done, in particular a new kernel space implementation as opposed to the previous user space implementation. By hosting iWarp in the kernel, we enable iWarp to be used as a system utility for kernel-resident clients such as network file systems and remote block devices. Having a kernel implementation also enables the possibility of fine tuning the TCP/IP stack for improved performance down the road. We are also able to eliminate as many data copies as possible while residing in the kernel. This kind of control over the operating system and access to its internal routines is not possible in a user space implementation.

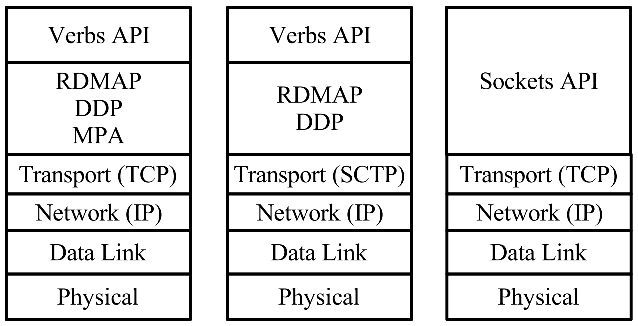


Figure 1. iWarp/TCP, iWarp/SCTP and sockets/TCP protocol stacks.

3 What is iWarp?

As mentioned in the previous sections, iWarp is comprised of three protocol layers built on top of TCP/IP, and a verbs/API layer that interfaces between client applications and the underlying protocol stack. A pictorial representation of the iWarp stack as compared to the traditional sockets over TCP stack is given in Figure 1. As indicated in the figure, it is also possible for iWarp to run over SCTP. Since SCTP provides a built in framing mechanism there is no MPA layer necessary, refer to 3.4 for more details on this.

3.1 Verbs or API

The very top layer of the stack is the verbs, or API layer. This is the only interface into the iWarp stack that is accessible to a user application. There is one possible specification available [9], but more than likely each vendor will produce their own API. As long as the API conforms to the verbs specification, then the implementation is compatible in semantics if not in syntax with other iWarp implementations. Having a verbs specification and a low level API is important for supporting higher level languages and programming paradigms such as MPI [13] and DAPL [8]. Unlike our iWarp API, the traditional sockets API does not provide the semantics necessary to implement zero-copy messaging.

3.2 RDMAP

The RDMA Protocol (RDMAP) layer is the portion of the stack that is responsible for coordinating RDMA data transfers. The RDMAP layer follows a very specific protocol, which is specified in [19]. In general, the initiation of an RDMA write, or send message is verified by this layer and passed down to the DDP layer. Messages received by lower layers are retrieved from the network in sections beginning with the message header. This header is presented to the RDMAP layer to determine the local destination for the payload. In the case of an RDMA write message, the local destination is specified in the message header itself. RDMAP verifies the incoming message parameters and returns buffer and length information to DDP. Untagged incoming messages simply consume the next buffer from a list of pre-posted receive buffers.

3.3 DDP

DDP, or Direct Data Placement [20], is mainly responsible for directly placing incoming data into user application buffers without making a copy. DDP does this by segmentation. On sends, DDP splits outgoing messages into chunks that will fit into lower-level transport frames. On the receiving end, DDP reassembles these frames and places the data at the appropriate offset in the destination buffer. DDP places a small header in each outgoing message to specify the queue number, message number and offset as well as an opaque field from RDMAP.

3.4 MPA

Since TCP is a byte-oriented streaming protocol, there is no guarantee that data will arrive in order on the receiver. This causes problems for DDP, in order to avoid buffering, and thus creating a copy of the incoming data, the Marker PDU Aligned Framing layer (MPA), is used [4]. MPA adds periodic markers to the byte stream to allow the remote side to rediscover the message boundaries. The other contribution is a stronger CRC-32 checksum to supplement the known insufficient TCP checksum [22]. As mentioned previously, when SCTP is the underlying transport layer, MPA is not needed. This is because SCTP already provides the necessary framing to rediscover message boundaries, as well as implements a stronger check sum.

3.5 Hardware Considerations

An important aspect to consider is the role that hardware plays. In traditional TCP processing, everything above the Data Link layer is done in software, and the CPU is responsible for moving data to and from the NIC, by going through the kernel where copies are made. In a TOE card, everything from the TCP layer down is done in hardware. Yet the CPU still has to move data to the NIC by passing it through the kernel and making copies along the way. With an iWarp RNIC the story is quite different. Everything from RDMAP down should be implemented in hardware. This means that the RNIC moves data directly into and out of application buffers, and not the CPU. The kernel is bypassed and no copies are required. The RNIC provides for much lower CPU utilization, and requires far less memory bandwidth than the traditional TCP mechanisms, and to a lesser extent the TOE card as well. It may seem counterintuitive that we are creating an iWarp stack to run solely in software, but one must keep in mind, the benefits outlined in this section will still be realized by the host which does have the RNIC—we do not expect to directly benefit the host without an RNIC. Although the non-accelerated host could possibly see an indirect benefit, in that the server with reduced system load could be able to handle more clients, or handle clients in a shorter amount of time.

4 Design and Implementation

Although much of the code in this work is similar in function to our previous user space work [5], there are numerous challenges that arose in implementing the stack in kernel space. We have tested our stack against the Ammasso 1100 RNIC [1] to ensure compliance with existing hardware. This required us to provide facilities to disable MPA since the Ammasso RNIC does not implement the specification fully. We have also added the option to disable CRC so that we can understand fully the impact of this CPU intensive operation.

4.1 About the code base

Our source code base for both user level and kernel iWarp is made up of roughly 20,000 lines of ANSI C code. It is known to build with GCC version 3. There are no special hardware or software requirements beyond, a 2.6 Linux kernel and GCC. While the user level stack should work on Linux with a 2.4 or earlier kernel (not tested), the kernel iWarp stack will only work on a 2.6 kernel, due to the different ways in which modules are handled between the two versions. It is worth noting, that our kernel module, does not require any kind of patch to the kernel. It can be loaded and unloaded at will if the running kernel supports it. Errors from the kernel stack are visible in `dmesg`. The code has been tested and verified to work on both 32 and 64 bit versions of Linux.

4.2 Kernel Implementation

In the kernel, we create a character device to serve as a software RNIC device. This enables user level interaction with the kernel-resident iWarp stack through ordinary `read` and `write` calls.

4.3 Verbs API

By using the character device and standard I/O interface we are able to still have our verbs API reside in user space, as in our previous work. While the API remains the same to the end user application, a complete rewrite of the verbs layer's internal interface with the iWarp stack was necessary. Care had to be taken so that the verbs API remains functional with both user and kernel level iWarp stacks. This means that an application can be run on either kernel or user level iWarp and no change in application code is needed. Just a recompile to pick up the appropriate libraries.

4.4 TCP Interface

Ideally various aspects of MPA would be integrated into the TCP stack itself. While this would be quite feasible, the changes would require modifications to the kernel beyond an independent software module and users would need a special kernel patch to use our module. To avoid this requirement we restricted our interactions with the kernel TCP stack to well-known exported interfaces, in particular, `kernel_sendmsg` and `kernel_recvmsg`. This choice brings two major implications: first, that the application is blocked while sending and, second, that message reception is driven by application polling. Due to the goals of software iWarp in reducing CPU utilization on the server or hardware side of the communication, these restrictions are in practice not bothersome on the client or software side.

4.5 Threading Model

One way to avoid the two limitations mentioned previously would be to allocate a separate kernel thread to manage outgoing sends and to block for incoming messages while the main application thread continues its work. This extension is not difficult and may benefit some applications that attempt to overlap communication and computation, or multiple workloads on the same machine. There is a trade off between simplicity of single-threaded model versus performance and complexity of multi-threaded implementation. For our kernel port we chose the former, as performance on the software side is not our focus.

4.6 Memory Registration

The application buffers must be pre-registered with the operating system before they are used for the transmission or reception of messages. Furthermore, due to segmentation of the the physical address space as used by the Linux kernel, pages must be mapped into the kernel address space using `kmap` and removed with `kunmap` before they can be used by the TCP stack. We use reference counting to map the pinned pages only when necessary. While pinning lasts for the duration of a memory registration, kernel mapping is restricted from the beginning of a message to its completion. Note that for machines with 64-bit address spaces, calls to `kmap` and `kunmap` are unnecessary and are optimized away because the kernel has enough address bits to map the entire physical memory at once. In spite of this, we use the mapping facility to support a broader range of architectures.

5 Results

We now present the results from experiments undertaken to explore performance and interoperability of our kernel software iWarp stack. The three key metrics which we will concentrate on are latency, bandwidth and CPU utilization, or system load. While the performance of our software stack is not our main goal, we must still verify that it does not cause a drastic drop in performance. The impact of the software stack must be small or else it will not find use in any real application. As mentioned throughout this paper, our main focus is on reducing the load on the system with the hardware iWarp when communicating with non-hardware equipped hosts.

The experiments are run between combinations of nodes using three different iWarp stacks: the kernel implementation presented in this paper, the user-space implementation we presented earlier, and the Ammasso hardware implementation. Our results demonstrate that all these stacks do inter-operate, but that performance varies.

5.1 Test Environment

The test environment for this work consists of a 71 node Linux cluster, of which 41 nodes are outfitted with an Ammasso 1100 RNIC [1]. Each node has two Opteron 250 processors with 2 GB RAM and an 80 GB SATA disk, although we disabled one processor to obtain more accurate utilization results. There are two Tigon 3 Gigabit Ethernet adapters built into the Tyan S2891 motherboard. The Ammasso RNICs and management networks are connected by two SMC switches. A switch was measured to contribute roughly 3.5 μs latency to the flight time of small packets, up to about 150 bytes, then a constant slope at line rate consistent with store-and-forward operation leading to a 14 μs delay at full 1500-byte frames.

5.2 Latency

We define latency as half of the total round-trip time of a ping-pong message sent from one host to another, and back. All measurements include operations by the user application to send outgoing messages, and detect reception of incoming ones. We use iWarp RDMA writes in all tests, and use socket `write` calls for TCP tests. It is worth keeping in mind that a software iWarp RDMA write is really a TCP socket `write` under the hood.

	4 byte messages	64 kB messages
hw-hw	16.1 ± 0.3	614.2 ± 3.3
ksw-hw	18.7 ± 0.2	619.7 ± 1.2
tcp-tcp	16.9 ± 0.2	594.8 ± 18.9

Table 1. Latency overview (μs).

In Table 1 we present the latency measurements for different node configurations. These numbers were gathered by connecting the respective hosts back to back, via a crossover cable, so we can discount the delay in latency introduced by the switches. The *hw-hw* line is two Ammasso RNICs and we can see a slightly lower latency for 4 byte messages than for the kernel software to hardware combination, *ksw-hw*. The values at the larger message size are roughly identical given the measurement errors. The surprising result in contrast to our previous work [5] is that the TCP latency, *tcp-tcp* is quite low. The difference is that the Tigon 3 NICs perform better than the Intel eepro1000 NICs used in that previous work, and that the processors in this test equipment are somewhat faster.

While our kernel implementation of software iWarp does add some latency to communication with a hardware device, it is rather small and at larger message sizes becomes negligible. It comes from the overhead of message processing in software on one side of the communication.

5.3 Throughput

In the case of throughput, we are measuring the time it takes to send many one-way messages from the source to the destination, then the return of a short message from the destination to the source to tell when all messages are received. As with the latency tests we used iWarp RDMA writes.

Unlike with latency, for throughput we must consider separately the case when software iWarp acts as the sender or the receiver side due to the fact that the load of sending a stream of messages can be quite different from receiving them. Figure 2(a) shows the throughput for stock TCP on both sides, and back-to-back Ammasso RNICs, as well as an Ammasso card talking to kernel software iWarp as both a sender and receiver, and a pure software situation.

The TCP curve is consistently above the others by about 10 Mb/s, but this extra performance comes at a cost in processor utilization as we shall see later. The pure software iWarp configuration, SW→SW, tracks the TCP curve in shape but lower by 10 Mb/s, while the curves involving a hardware Ammasso RNIC show a slight performance degradation at small message sizes, especially when hardware is the sender. This may be due to the apparent complexity involved in transmitting messages in the Ammasso RNIC implementation as the next section will show.

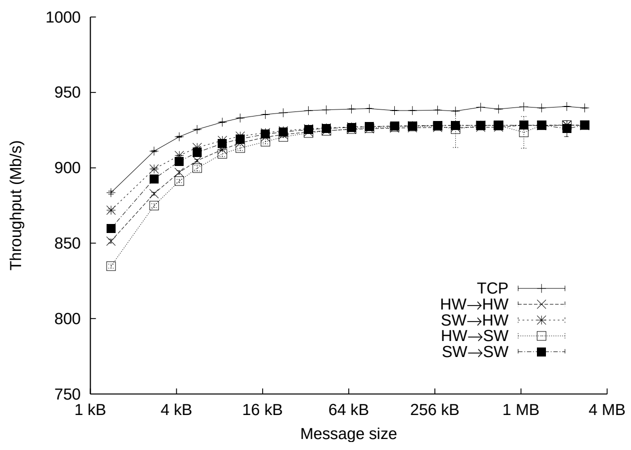
5.4 Comparison Against User Space Stack

We measured the differences between this kernel space implementation and our previous work in implementing an iWarp stack in user space [5]. Throughput results are shown in Figure 2(b) where there are two curves for each implementation, one with the CRC calculation and verification included and another with CRC disabled. The kernel and user space versions are essentially identical, and both suffer only a small (8 Mb/s) performance penalty due to the presence of the CRC. Again, in contrast to our previous work, the effect of the CRC on performance is very slight perhaps due to the use of more efficient host processors in this work.

	4 B messages	64 kB messages
kernel with CRC	20.3 ± 0.2	615.5 ± 1.2
user with CRC	19.6 ± 0.2	612.3 ± 1.9
kernel without CRC	20.1 ± 0.2	604.5 ± 0.8
user without CRC	19.5 ± 0.2	602.7 ± 0.8

Table 2. User vs kernel space latency (μs).

Turning our attention to latency, the first two lines of Table 2 compare the two implementations with CRC enabled, as is required by the MPA specification. There is a small (4%) latency penalty to using the kernel iWarp stack. Possible reasons for this could be that the completion queues live in kernel space, or the need to use `kmap` and `kunmap` to access user data before each send or receive operation. For completeness we also compare the two versions with CRC disabled and see no effect except a throughput-related degradation for large messages.



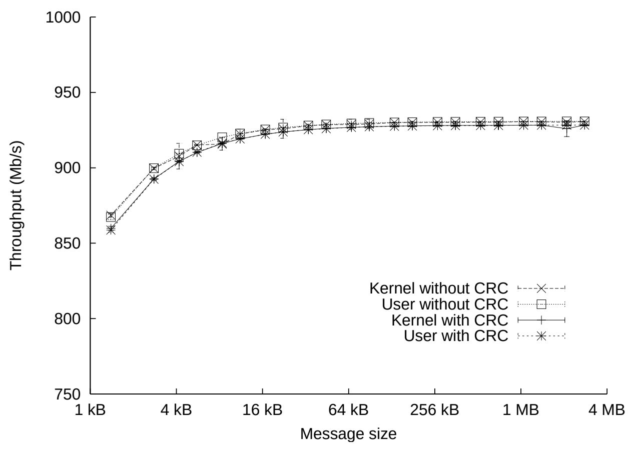
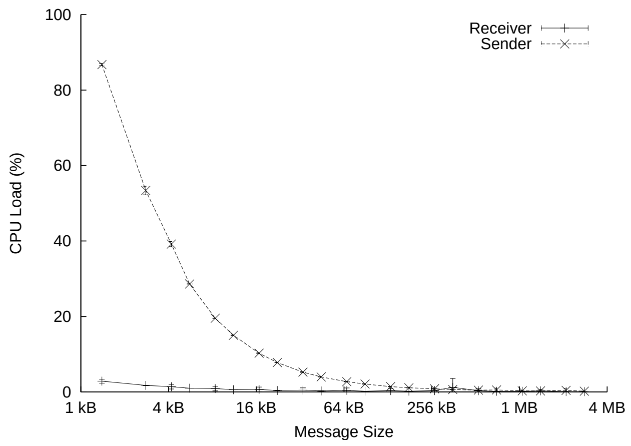


Figure 2. (a) Throughput overview. (b) User space vs. kernel space throughput.



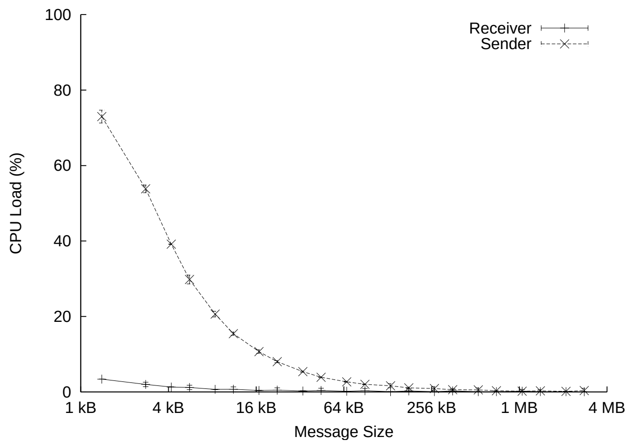


Figure 3. Host CPU utilization for a node using the hardware iWarp stack when connected to a peer with either (a) hardware or (b) software iWarp stack.

5.5 CPU Utilization

The main goal of our work, and one of the key reasons to adopt the iWarp protocol is the benefit of reduced CPU utilization. We show the percentage of the single host processor consumed by networking load for various configurations in Figures 3 and 4 as a function of message size.

An brief explanation of the figures aids in understanding what is going on. In figures 3 (a) and (b), what is shown is the percentage of CPU being used by a host equipped with a hardware iWarp card. In 3(a) we show what happens when the other host is equipped with a hardware iWarp adapter, and in (b) we show what happens when the other host is running our software iWarp stack only. In Figure 4(a) we show the CPU usage of a host communicating with another host using only standard TCP sockets, as both a sender and receiver. Lastly in Figure 4(b) we have the CPU utilization for a host using our kernel software iWarp stack as both a sender and receiver, and communicating with a hardware iWarp equipped host. The difference between Figure 4(b) and 3(b) is that in 3(b) we are looking at the CPU usage on the hardware iWarp end, where as in 4(b) we are looking at CPU usage on the software iWarp end.

This data was collected by running our throughput tests while a low-priority program measured available CPU cycles and reported the load periodically [12]. Error measurements show the variation in load during each approximately 10 second throughput test. Because the throughput achieved by all the configurations are essentially the same as seen in Figure 2(a), direct comparisons of the processor utilization curves is valid.

We see from looking at the figures that the CPU utilization of iWarp is generally much lower than for TCP due to the offloading of much of the protocol work to the iWarp RNIC. It appears that sending small messages is a CPU intensive task for this particular iWarp adapter, but this a bit misleading. What is going on is that the hardware iWarp adapter is doing more work, sending more messages in a

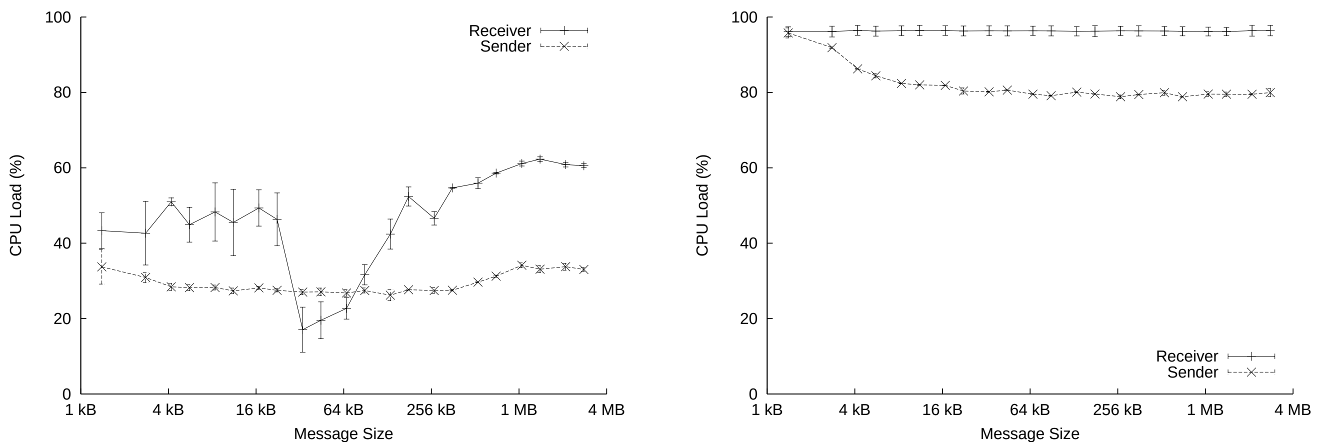


Figure 4. (a) Host CPU utilization, both nodes using TCP, not iWarp. (b) Host CPU utilization for a node using the software iWarp stack when connected to a peer using the hardware iWarp stack.

shorter amount of time. As for the "dip" in load in the TCP receiver curve, we have no explanation, and have found that this is a repeatable result.

As mentioned previously Figure 3(b) shows the CPU utilization experienced by a host that has a hardware iWarp RNIC but is communicating to another host that is using our kernel space software iWarp stack. These curves are almost identical to those in Figure 3(a) and supply the key results of this work: the host with the hardware iWarp RNIC can now take advantage of its protocol offload and zero-copy benefits due to the software iWarp protocol used by its peer.

In contrast, Figure 4(b) shows the near full processor load required by the software iWarp implementation. While not shown here, much of this load comes from the CRC-32c calculation as required by the MPA specification to support TCP networks. Without the CRC these curves more closely resemble the TCP curves in Figure 4(a).

6 Related and Future Work

Work related to zero-copy, OS bypass, and RDMA in general has been quite extensive in the past decade as problems with traditional TCP networking continue to surface. The performance of the Ammasso 1100 RNIC has also been the subject of at least one paper [7].

One project which somewhat parallels our work is another software implementation of iWarp [2], but takes a significantly different approach to the user interaction and is not compatible with existing iWarp hardware. The goal of [2] is to enable any sockets program (one that uses `read` and `write` function calls) to run over iWarp. While admirable from the point of compatibility with an existing code base, the performance of this approach is necessarily limited and we prefer to consider future applications that will target an RDMA-capable API.

In addition to work on the underlying transport, there has been much activity in application programming interfaces. Some well known examples are the DAT Collaboratives, DAPL [8], Mellanox's VAPI [10], a new API from the OpenIB Alliance [15], and of course the RDMA Consortium's iWarp verb specification [9].

In the future we plan to investigate a multi-threaded implementation of the iWarp software stack, and port some kernel-resident applications such as NFS. Integrating MPA into TCP may simplify the in-kernel interfaces somewhat as well. We have also implemented a file transfer utility and are pursuing a wide-area evaluation of iWarp. At this time we are able to confirm that our software iWarp stack is functional over the WAN when interoperating with an Ammasso iWarp adapter on the remote end. Performance analysis of iWarp at larger scale than presented in [7] is also currently underway.

7 Conclusion

To summarize we have demonstrated a software kernel-based iWarp stack that will enable a remote host to take advantage of its hardware iWarp. Our main contribution is in implementing and demonstrating the interoperability of a kernel-based software iWarp implementation. The results show the advantages of single-sided acceleration where hardware-enabled servers can sustain their performance with all the advantages of zero-copy and CPU offload even when clients are not similarly equipped. Not only user-space but kernel-space applications like NFS will now be able to take advantage of iWarp.

In the near future we expect to see more 10 Gb/s networks and iWarp adapters come into use. While a modern CPU can keep up with the demands of TCP processing on a 1 Gb/s network, processor utilization to feed faster networks will soon become more important.

The work presented here should further more widespread acceptance of iWarp and for other future research into iWarp and related protocols. The software discussed is freely available [6] for use under the GNU General Public License.

Acknowledgment

We heartily thank the reviewers for their thoughtful comments.

References

[1] Ammasso Inc. Ammasso 1100 product description. http://www.ammasso.com/products.htm, 2005.

[2] P. Balaji, H.-W. Jin, K. Vaidyanathan, and D. Panda. Supporting iWARP compatibility and features for regular network adapter. In *Proceedings of the IEEE Cluster 2005 Conference, RAIT Workshop*, 2005.

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

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

[5] D. Dalessandro, A. Devulapalli, and P. Wyckoff. Design and implementation of the iWarp protocol in software. In *Proceedings of PDCS'05*, Phoenix, AZ, Nov. 2005.

[6] D. Dalessandro, A. Devulapalli, and P. Wyckoff. Software iWarp user and kernel software distribution. http://www.osc.edu/research/network_file/projects/iwarp/index.shtml, 2005.

[7] D. Dalessandro and P. Wyckoff. A performance analysis of the Ammasso RDMA enabled ethernet adapter and its iWARP API. In *Proceedings of RAIT'05 in conjunction with Cluster'05*, Sept. 2005.

[8] DAT Collaborative. Direct access transport. http://www.datcollaborative.org.

[9] J. Hilland, P. Culley, J. Pinkerton, and R. Recio. RDMA Protocol Verbs Specification. http://www.rdmaconsortium.org/home/draft-hilland-iwarp-verbs-v1.0-RDMAC.pdf, April 2003.

[10] InfiniBand Trade Association. *InfiniBand Architecture Specification*, Oct. 2004.

[11] C. Kurmann, M. Muller, F. Rauch, and T. Stricker. Speculative defragmentation: A technique to improve the communication software efficiency for gigabit ethernet. In *HPDC 2000*, 2000.

[12] A. Morton. Cyclesoak. http://www.zipworld.com.au/~akpm/linux/#zc.

[13] MPI Forum. *MPI: A Message-Passing Interface Standard*, Mar. 1994.

[14] Myricom. Myrinet. http://www.myri.com/.

[15] Open Infiniband Alliance. OpenIB. http://www.openib.org/.

[16] Open Infiniband Alliance. OpenIB-iWarp. http://www.openib.org/docs/pr071305.doc.

[17] J. Pinkerton, E. Deleganes, and S. Bitan. DDP/RDMAP security. http://www.ietf.org/internet-drafts/draft-ietf-rddp-security-07.txt, Apr. 2005.

[18] Quadrics. http://doc.quadrics.com.

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

[20] H. Shah, J. Pinkerton, R. Recio, and P. Culley. Direct data placement over reliable transports. http://www.ietf.org/internet-drafts/draft-ietf-rddp-ddp-04.txt, Feb. 2005.

[21] P. Shivam, P. Wyckoff, and D. K. Panda. EMP: zero-copy OS-bypass NIC-driven gigabit ethernet message passing. In *Proceedings of SC'01*, Denver, CO, Nov. 2001.

[22] J. Stone and C. Partridge. When the CRC and TCP checksum disagree. *ACM Sigcomm*, Sept. 2000.

[23] 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, Dec. 1995.