# Analyzing the Impact of Supporting Out-of-Order Communication on In-order Performance with iWARP[*]

P. Balaji[†] W. Feng[§] S. Bhagvat[‡] D. K. Panda[λ] R. Thakur[†] W. Gropp[†]

[†]Mathematics and Computer Science,
Argonne National Laboratory
{balaji, thakur, gropp}@mcs.anl.gov

[§]Department of Computer Science,
Virginia Tech
feng@cs.vt.edu

[‡]Scalable Systems Group,
Dell Inc.
sitha_bhagvat@dell.com

[λ]Computer Science and Engineering,
Ohio State University
panda@cse.ohio-state.edu

**Abstract**

Due to the growing need to tolerate network faults and congestion in high-end computing systems, supporting multiple network communication paths is becoming increasingly important. However, multi-path communication comes with the disadvantage of having to deal with the out-of-order arrival of packets (because packets may traverse different paths). While modern networking stacks such as the Internet Wide-Area RDMA Protocol (iWARP) over 10-Gigabit Ethernet (10GE) support multi-path communication, they do not handle out-of-order packets primarily owing to the overhead on in-order communication that it adds. Specifically, in iWARP, supporting out-of-order packets requires *every* packet to carry additional information causing significant overhead on packets that arrive in-order. Thus, in this paper, we analyze the trade-offs in designing a feature-complete iWARP stack, i.e., one that provides support for out-of-order arriving packets, and thus, multi-path systems, while focusing on the performance of in-order communication. We propose three feature-complete designs of iWARP and analyze the pros and cons of each of these designs using performance experiments based on several micro-benchmarks as well as an iso-surface visual rendering application. Our analysis reveals that the iWARP design providing the best overall performance depends on the particular characteristics of the upper layers and that different designs are optimal based on the metric of interest.

## 1 Introduction

As high-end computing (HEC) systems continue to increase rapidly in size, their network subsystems must scale as well, particularly to address issues such as hot-spot congestion [27, 31] and hardware faults [8]. Multi-path

[*]This research is funded in part by DOE Grant #DE-FC02-06ER25755, NSF grants #CNS-0403342, #CNS-0509452 and #CPA-0702182 and by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

communication, supported by InfiniBand (IB) [1] and 10-Gigabit Ethernet (10GE) [18], provides a way to address these issues. In IB, a subnet manager [32] tracks the topology of the system and provides capabilities for querying and setting up multiple paths for a connection. In contrast, 10GE uses VLAN-based multi-pathing [29, 25] – a technique that allows the system to build multiple overlay tree structures on the same physical network.

While multi-path communication has its advantages, it also possesses the disadvantage of out-of-order arrival of packets. That is, because the packets of a given connection can be sent over different paths, packets injected into the network later might arrive at the destination before packets that were injected into the network earlier. Unfortunately, current networks do not deal with this issue, leaving it instead to the protocol stacks of these networks to handle it. The communication protocol of IB takes a simplistic approach to the problem by dropping out-of-order packets and relying on the sender to retransmit them. This, of course, can lead to significant performance degradation in large-scale HEC systems. In contrast, for the 10GE network, the Internet Wide-Area RDMA Protocol (iWARP) [2] specifies a more elegant way to deal with out-of-order packet arrival; it directly places out-of-order packets at the appropriate location in the destination buffer, while delaying informing the application about the message till all packets corresponding to this and all previous messages have arrived.

The iWARP stack (Figure 1) is a new initiative by the Internet Engineering Task Force (IETF) and RDMA Consortium (RDMAC) to provide capabilities such as remote direct-memory access (RDMA) and zero-copy data transfer. iWARP maintains compatibility with the existing TCP/IP infrastructure by stuffing iWARP frames within TCP/IP packets. On the sender side, each TCP/IP packet contains a single iWARP frame. However, intermediate switches, such as those which support splicing [11], can segment a single packet into multiple packets or coalesce multiple packets into a single packet (Figure 2). Thus, if the first packet is delayed, the later-arriving packets could either contain a complete iWARP frame or a part of it. However, without additional information, the receiver cannot determine which packets contain a full iWARP frame and which packets do not. Even if no segmentation occurs and all the packets contain full iWARP frames, the receiver has no way to determine this and must assume the possibility of segmentation for all packets.

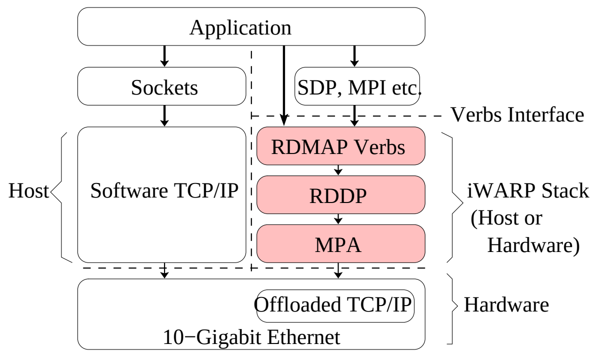


Figure 1: iWARP Protocol Stack

The iWARP standard specifies a solution to handle such scenarios by providing each packet with additional

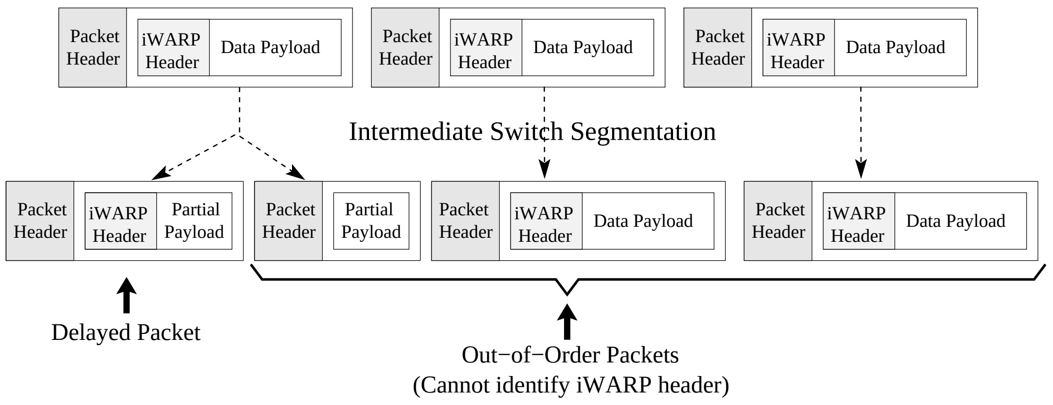


Figure 2: Out-of-Order Arrival of Packets

information, so as to allow the receiver to correctly determine the iWARP frames within TCP packets (handled as a part of the MPA layer in Figure 1). However, adding such additional information complicates the packet structure and processing required, thus impacting the performance of in-order communication. As a result, currently available implementations of the iWARP standard [21, 19] do not provide any capabilities to handle out-of-order arriving packets and follow simplistic approaches such as those used by IB.

To address this shortfall, in this paper we analyze the trade-offs in designing a feature-complete iWARP stack, i.e., one which provides support for out-of-order arriving packets, and thus, multi-path communication, while focusing on the performance of in-order communication. With the added complexity associated with feature-complete iWARP designs, offloading the entire iWARP stack onto the network interface card (NIC) may not always be beneficial. Accordingly, we study the trade-offs by analyzing three different feature-complete designs of iWARP over 10GE NICs with hardware implementations of TCP/IP [14], namely (i) host-based iWARP, which is completely in host space, (ii) host-offloaded iWARP, where iWARP is completely offloaded from the host onto the NIC, and (iii) host-assisted iWARP, where iWARP is only partially offloaded onto the NIC. Our analysis reveals that the iWARP design providing the best overall performance depends on the particular characteristics of the upper layers and that different designs are optimal based on the metric of interest.

Furthermore, in order to demonstrate the capability of each iWARP design as applicable to the broader research community, we also evaluate an iso-surface visual rendering application [7] using the three iWARP implementations. This application uses iso-surface rendering techniques to simplify the visual representation of large datasets such as those corresponding to oil reservoirs and biomedical virtual images. Though the application was initially designed on top of TCP/IP sockets, we modified it to directly utilize the native iWARP verbs interface so that it can be evaluated with the different iWARP designs proposed in this paper. Our analysis shows that depending on subtle changes in the application parameters, e.g., the granularity of data distribution, either the host-offloaded iWARP or the host-assisted iWARP can provide the best performance.

The remaining part of the paper is organized as follows. In section 2, we present a brief overview of the iWARP stack. We describe the various design choices for a feature-complete iWARP implementation in section 3. Performance results are presented in section 4. We describe prior literature related to our work in section 5 and conclude the paper in section 6.

## 2 Overview of the iWARP Standard

iWARP comprises of three protocol layers atop a reliable IP-based protocol such as TCP: (i) RDMAP verbs, (ii) Remote Direct Data Placement (RDDP) protocol and (iii) Marker PDU Aligned (MPA) protocol. RDMAP verbs is a thin interface which allows applications to interact with RDDP. In this section, we describe the details about RDDP and MPA, that are relevant to this paper. More details about these and RDMAP verbs are in [2].

**Remote Direct Data Placement (RDDP) Protocol:** RDDP provides the core of the data communication processing in the iWARP stack. It aims at providing both channel based semantics (i.e., send/receive communication) as well as memory based semantics (i.e., RDMA communication) to its upper-layer protocols (ULPs). RDDP provides reliable, in-order delivery using a reliable IP based protocol such as TCP. RDDP distinguishes iWARP from other high-speed network stacks based on its capability to decouple data placement and message delivery, i.e., even if packets arrive out-of-order, RDDP directly places them in the appropriate location of the final destination buffer (data placement); however, the upper-layer is informed about the placement of the data only after the entire message is placed (data delivery). This, of course, assumes that RDDP can correctly identify and understand the contents of out-of-order TCP/IP packets. The Marker PDU Aligned (MPA) protocol provides RDDP with the necessary support for achieving this.

**Marker PDU Aligned (MPA) Protocol:** RDDP has several limitations. First, it is an end-node protocol; it need not be supported by intermediate nodes. For switches that support splicing [11] (e.g., firewalls and port-forwarding switches), this leads to *middle box fragmentation*, i.e., packets going into the switch can be segmented into multiple packets or multiple packets can be coalesced into a single packet. This makes it impossible for

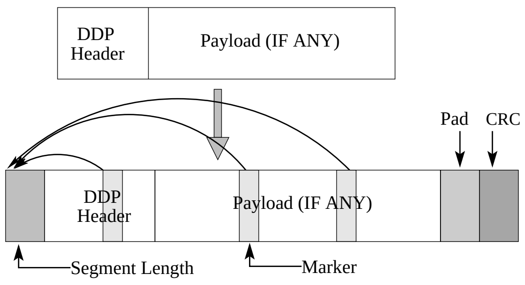


Figure 3: MPA Protocol Frame

the end node to recognize the RDDP headers without additional information, if packets arrive out-of-order. Second, the data-integrity check performed by TCP/IP (i.e., checksum) has been shown to be error prone in many cases [30]. Accordingly, several upper layers perform additional data integrity checks such as the Cyclic Redundancy Check (CRC).

In order to tackle these problems, iWARP uses MPA [12]. Figure 3 illustrates the new iWARP frame format with MPA, known as the Framing Protocol Data Unit (FPDU). The FPDU format has three essential changes. First, it introduces strips of data, known as *markers* pointing to the RDDP header. These are spaced uniformly based on the TCP sequence number and provide the receiver with a deterministic way to find them. When a packet arrives out-of-order, it can use these markers to identify where the *start* of the iWARP frame is; once the start is identified the iWARP headers can be recognized, and the remaining fields of the frame, such as the packet length, can be acquired. Thus, this allows iWARP to identify if that particular packet contains a complete iWARP frame or a partial one. Second, MPA uses a 32-bit CRC check together with any other data integrity check provided by the underlying protocols. Third, each frame is padded with up to three pad bytes so as to ensure that it is of a length which is a multiple of four bytes. Of these, CRC is easily the most compute intensive. The placement of the markers, on the other hand, is a tricky operation. Since the markers are placed in between the data stream, the data has to be moved in order to do this. There are a number of ways of doing this, as we will see in section 3, each having its pros and cons.

## 3 Design Choices for iWARP

As described in section 1, designing a feature-complete iWARP stack, i.e., one which provides support for out-of-order arriving packets, is a non-trivial task. Specifically, as discussed in section 2, such capability makes the packet format and processing significantly more complex. In this section, we describe three different design choices for handling such additional complexity: (i) host-based iWARP (in section 3.1), (ii) host-offloaded iWARP (in section 3.2) and (iii) host-assisted iWARP (in section 3.3).

In practice, a complete spectrum of design choices exist for implementing feature-complete iWARP. For example, with host-offloaded iWARP, depending on the hardware features provided by the NIC, a large number of different design choices exist. Similarly, with host-assisted iWARP, design choices exist with respect to what components can be offloaded to the NIC and what components be retained at the host. However, dealing with all these choices is outside the scope of this paper. Instead, we pick two design choices for host-offloaded iWARP based on the NIC hardware components that are widely available in the commercial market, and one design choice for host-assisted iWARP based on our understanding of the computational complexity of the iWARP stack, and analyze them.

For each of these designs, amongst the various tasks that need to be handled within the iWARP stack, we identify that three tasks are of particular importance: (i) CRC based data-integrity, (ii) connection demultiplexing and (iii) placement of markers. We describe the pros and cons of each iWARP design based on these three tasks.

## 3.1 Host-based iWARP

*Host-based iWARP* is a completely software-based design of iWARP that has been proposed and implemented by several researchers earlier [4, 13]. It is a generic design that can be used on any Ethernet adapter while maintaining complete compatibility with hardware implementations of iWARP. In this section, we summarize some details of this design.

There exist two designs for host-based iWARP, one in user-space and one in kernel-space. The user-space design builds the iWARP stack on top of TCP/IP sockets. Aspects such as asynchronous communication are handled using a separate thread. The kernel-space design, on the other hand, bypasses TCP/IP sockets and allows applications to directly communicate with the iWARP stack. In this paper, we only deal with the kernel-level host-based iWARP because of its better performance, and describe the pros and cons of its design based on the three tasks in the iWARP stack which we identified earlier, i.e., CRC based data-integrity, connection demultiplexing, and placement of markers.

**CRC based data-integrity:** CRC is one of the most compute intensive tasks in the iWARP stack. There have been several attempts to improve its performance [28, 9], often at the cost of additional memory usage. However, its computational overhead is still considered to be very high [22]. Thus, since CRC is performed by the host, it accounts for a significant overhead in this design.

**Connection Demultiplexing (DEMUX):** Traditional TCP/IP performs demultiplexing (DEMUX) of packets in host-space, i.e., the NIC hands over all packets to the host and the host identifies the connection to which each packet belongs and places it in the appropriate queue. While this is not a major concern for applications that only deal with a single (active) connection, this introduces significant overheads for applications dealing with several connections simultaneously (e.g., cache thrashing and CPU interruption for non-critical data). Host-based iWARP uses the DEMUX done by TCP/IP in host-space, resulting in a high overhead.

**Placement of Markers:** In order to deal with out-of-order placement of data, iWARP inserts *markers* at regular intervals in the data stream. These markers point to the RDDP header of each iWARP frame allowing the receiver to recognize the frame boundaries. Since markers are inserted within the data stream (Figure 3), an additional copy of data is required to allow this in this design, which adds more overhead.

In summary, all of the above three tasks add significant overheads for the host-based iWARP design. The only advantage of this design is its generality, which allows it to be utilized on any NIC irrespective of its capabilities.

## 3.2 Host-offloaded iWARP

*Host-offloaded iWARP* is completely offloaded from the host and implemented on the hardware and firmware present on the NIC, specifically taking advantage of the various hardware engines such as the CRC engine, DEMUX engine and DMA engines. This approach is similar to that taken by most modern cluster interconnects such as IB and Quadrics [26]. In this section, we point out aspects that make this design different for iWARP as compared to other networks.

For the first two tasks (CRC and DEMUX), the use of hardware engines allows host-offloaded iWARP to reduce the computational requirements of the host and improve performance. The capabilities of these hardware engines is similar to that of other networks (e.g., IB, Quadrics). The third task (placement of markers), however, is tricky. As described in section 2, placement of markers is done in between the data stream. Thus, the data has to be split to insert these markers, i.e., it needs to be placed in a non-contiguous manner on the NIC memory. Also, though the data at the host that needs to be transmitted by the application might be contiguous in virtual address space, it is quite likely that it will be non-contiguous in physical address space (split as physical pages). Thus, the ideal approach to insert markers would be to use a hardware DMA engine that is capable of *true* scatter/gather DMA, i.e., capable of transferring a set of noncontiguous buffers on the host memory to a set of noncontiguous buffers on the NIC memory in a single DMA operation. However, this is not easy to achieve.

Though several NICs provide an API for scatter/gather DMA, the operation itself is implemented as multiple independent DMA operations. The API is really more of a programming convenience. Some networks such as Quadrics optimize scatter/gather DMA operations by using DMA chaining (pipelining multiple DMA operations); but this still does not get rid of the requirement for multiple DMA operations. Though, in theory, it is possible to build a custom DMA engine to perform scatter for DMA read and gather for DMA write (generic scatter/gather is not possible due to chipset restrictions), no commercially available DMA engine currently does that because of the complexity it introduces. Thus, based on the capabilities of current DMA engines, we pro-

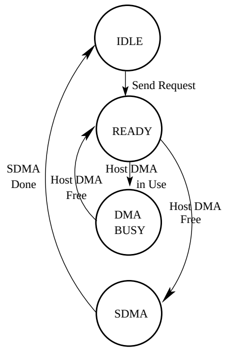


Figure 4: SDMA

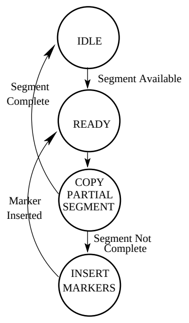


Figure 5: Processing

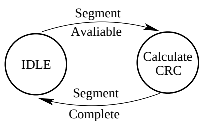


Figure 6: CRC

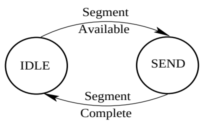


Figure 7: SEND

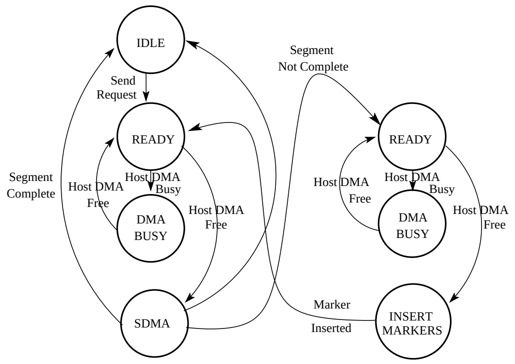


Figure 8: Integrated

pose two design variants for host-offloaded iWARP: (i) Contiguous DMA and decoupled marker insertion and (ii) Scatter/Gather DMA based marker insertion.

**Contiguous DMA and Decoupled Marker Insertion:** In this design, the NIC DMAs a sufficiently large contiguous chunk of the data (e.g., 2KB) and moves the data on the NIC to insert markers within the data stream. In other words, the DMA operation is decoupled with marker insertion. The advantage of this approach is that large contiguous data segments can be fetched in each DMA operation; thus the number of DMAs are less and the performance high. The disadvantage is that data has to be moved on the NIC in order to insert the markers.

To understand the disadvantages of this approach more clearly, we modeled the NIC processing in software using 4 threads (in each communication direction), each performing the tasks of the different processing engines on the NIC, namely: (i) Send DMA (SDMA) engine (RDMA engine for receive communication), (ii) Processing engine, (iii) CRC engine and (iv) SEND engine (RECV engine for receive communication). The states taken by each of these threads are represented in Figures 4-7. Note that this model only represents the processing and memory overheads with this design and does not simulate the actual performance of the approach.

In the model illustrated, the main processing of the SDMA engine is the DMA of data from host to NIC memory, thus touching NIC memory once for each byte transferred over the network. The processing engine's functionality is more complex; this engine has to move data that is DMA'ed by the SDMA engine to a different memory location while creating splits within the data stream so that the markers can be placed in these splits. This engine touches NIC memory twice (one to read the data DMA'ed by the SDMA engine and once to write it in the split format) for every byte transferred. The CRC engine and the SEND engine (to transfer data from NIC memory to the wire) touch NIC memory once each, summing up to five memory transactions for every byte transferred over the network. A number of NICs available in the market allow a single engine to process CRC while simultaneously transferring data to the wire; this reduces the number of memory transactions to four, which is still very high.

**Scatter/Gather DMA based Marker Insertion:** This approach is similar to the previous approach, except that the functionality of the SDMA engine and the processing engine are integrated (Figure 8). This integrated engine DMAs small noncontiguous chunks of data (represented by a scatter/gather list) and directly places them with markers within the data stream. The advantage of this approach is that it cuts down two NIC memory transactions. The disadvantage is that, as mentioned earlier, scatter/gather DMAs are implemented as multiple independent DMA operations by most NICs. Therefore, in this approach, the DMA operations are for small chunks of data (the iWARP standard requires the marker separation to be 512 bytes). Thus, the number of DMA operations is high, making it less efficient.

From implementation perspective, only the second approach was possible on the NICs we used. We implemented this using the reference iWARP code from Chelsio. However, due to the limited programmability of the NIC, we had to generate the scatter/gather list (corresponding to the data segments and the markers) on the host for each MTU chunk before handing it over to the NIC through multiple explicit communication calls. In other words, if a 2KB data message needs to be sent out with a marker separation of 512 bytes, we post a scatter/gather request with multiple (5-7) different entries, each pointing to appropriate sized data segments and markers.

## 3.3 Host-assisted iWARP

*Host-assisted iWARP* is a hybrid design which takes the best features of host-based iWARP and host-offloaded iWARP. Specifically, host-assisted iWARP performs compute intensive tasks such as CRC and DEMUX completely on the NIC using dedicated processing engines, while retaining tasks such as marker insertion on the host. This means that the copy required to insert markers on the host is the only overhead associated with this design. The advantage of such partial offload is that it efficiently utilizes the capabilities of both the NIC as well as the host. It is to be noted that this approach primarily focuses on raw performance – other metrics such as CPU utilization, however, suffer compared to host-offloaded iWARP due to the involvement of the host in the protocol processing tasks.

Figure 9 and Table 1 summarize the tasks breakup for the three different designs of iWARP, i.e., host-based iWARP does everything on the host, host-offloaded iWARP does everything on the NIC and host-assisted iWARP does the marker insertion on the host and everything else on the NIC.

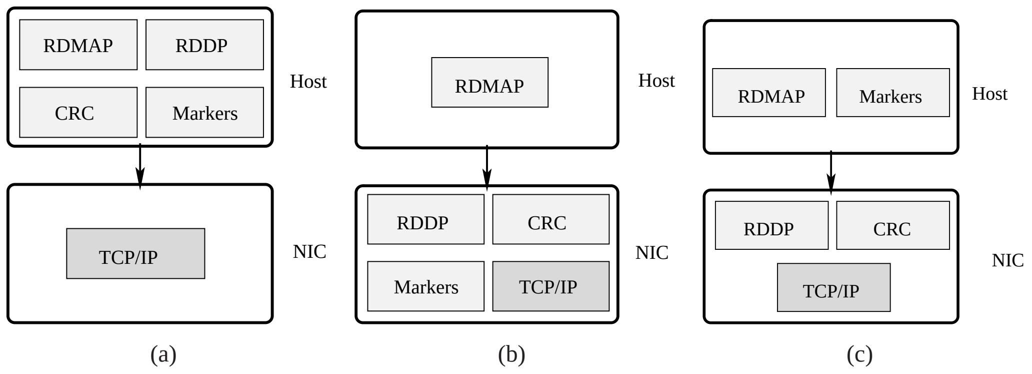


Figure 9: Task distribution for different iWARP designs: (a) Host-based, (b) Host-offloaded and (c) Host-assisted.

Table 1: iWARP designs

| | Host-based | Host-offloaded | Host-assisted | Comments |
|---|---|---|---|---|
| Insert Markers | Host | NIC | Host | Host-offloaded design requires multiple DMAs where as host-based and host-assisted designs required an extra copy. |
| CRC | Host | NIC | NIC | Host-offloaded design utilizes an additional CRC engine and hence reduces the host overheads seen in host-based iWARP. |
| DEMUX | Host | NIC | NIC | Host-offloaded design utilizes an additional DEMUX engine. |

## 4 Performance Results

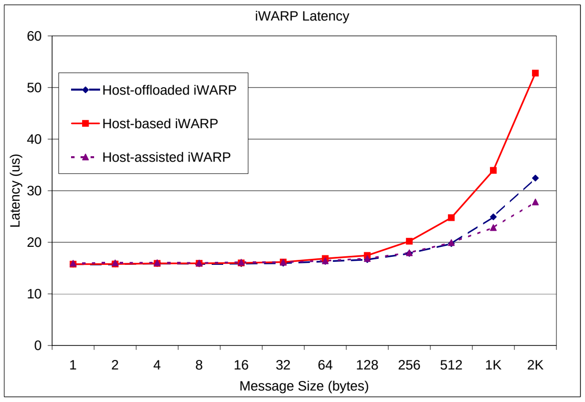
In this section, we evaluate the performance of the different iWARP designs using various micro-benchmark tests (section 4.2) as well as a iso-surface visual rendering application (section 4.3).

### 4.1 Experimental Testbed

For our experiments, we used a 4-node cluster built around SuperMicro SUPER X5DL8-GG motherboards with ServerWorks GC LE chipsets, which include 133-MHz PCI-X interfaces. Each node has two Intel Xeon 3.0 GHz processors with a 512-KB cache, a 533 MHz front-side bus and 2 GB of 266-MHz DDR SDRAM. The nodes are connected with Chelsio T110 10GE TCP offload engines through a 12-port Fujitsu XG800 switch. The software stack on the machines is based on linux-2.4.22smp and RedHat linux distribution. The driver version on the NICs is 1.2.0. For each experiment, ten or more runs/executions are conducted, the highest and lowest values dropped (to discard anomalies) and the average of the remaining values is reported. For micro-benchmark evaluations, the results of each run are an average of 10,000 or more iterations.

### 4.2 iWARP Evaluation

In this section, we evaluate the performance of the iWARP stacks using different micro-benchmarks.



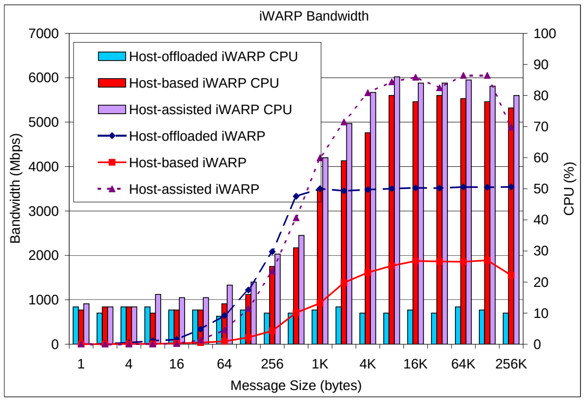


Figure 10: iWARP Micro-benchmarks: (a) Latency (b) Bandwidth

### 4.2.1 iWARP Latency, Bandwidth and CPU Usage

**Ping-pong Latency:** Figure 10(a) compares the ping-pong latency of the three designs. In this experiment, the sender sends a message of size *S* to the receiver. On receiving this message, the receiver sends back another message of the same size to the sender. This is repeated several times and the total time averaged over the number of iterations – this gives the average round trip time. The ping-pong latency reported here is one half of the round trip time, i.e., the time taken for a message to be transferred from one node to another.

As shown in the figure, for small messages, all three schemes perform similarly at about 16$\mu$s. As the message size increases, the performance of host-based iWARP deteriorates faster compared to the other designs. This is expected as this design does not take advantage of any advanced hardware present on the NIC. Comparing host-offloaded and host-assisted iWARP, as message size increases, host-assisted iWARP performs the best, outperforming host-offloaded iWARP by 10-15%. This trend is attributed to the overhead of multiple DMAs on host-offloaded iWARP, e.g., for a 2KB message, the host-assisted iWARP performs just one DMA operations, while with a marker separation of 512 bytes, host-offloaded iWARP needs to perform 5-7 DMA operations.

**Uni-directional Bandwidth:** Figure 10(b) shows a comparison of the uni-directional bandwidth. In this experiment, the sender sends a single message of size *S* a number of times to the receiver. On receiving all the messages, the receiver sends back one small message to the sender informing that it has received the messages. The sender calculates the total time, subtracts the one way latency of the message sent by the receiver, and based on the remaining time calculates the amount of data it had transmitted per unit time.

The basic trend for this result is quite similar to ping-pong latency, i.e., host-based iWARP performs the worst achieving a bandwidth of only about 2Gbps. Comparing host-offloaded and host-assisted iWARP, for very small messages host-offloaded iWARP performs slightly better; we are currently analyzing this behavior to understand its reasoning. For the peak bandwidth, however, host-assisted iWARP outperforms the host-offloaded iWARP with bandwidths of about 6Gbps and 3.5Gbps, respectively. The reason for the performance limitation of host-offloaded iWARP is again the number of DMA operations. Though, pipelining the DMA operations can improve the performance a little, it is eventually limited by the DMA overhead. Host-assisted iWARP, on the other hand, can DMA full length 2KB data chunks in each operation and thus can perform better.

**CPU Utilization:** Figure 10(b) also shows the CPU utilized by the different iWARP designs. To calculate the CPU utilization, we first run the experiment with a large number of iterations. When the experiment is running, on the same machine we take several sample measurements of the percentage of CPU cycles that are being used by the system (from `/proc`) and report the average.

As shown in the figure, host-offloaded iWARP uses the least amount of host-CPU (less than 15%) for all message sizes. As shown in prior research [5], as the message size increases, the percentage CPU used by hardware offloaded protocols should drop to zero, since most of the time in communicating large messages is spent on the network and does not use CPU. However, for the host-offloaded iWARP design, we notice that even for very large messages, the CPU overhead is constant and does not drop to zero. The reason for this counter intuitive behavior is attributed to our implementation of this design. Specifically, as mentioned in section 3.2, due to the limited programmability of the NICs we used, we had to generate the scatter/gather list (corresponding to the data segments and the markers) on the host before handing it over to the NIC through explicit communication calls. This causes the amount of host processing required in our implementation of the host-offloaded iWARP design, to linearly increase with message size and the percentage CPU used per unit time to be about constant.

The high CPU utilization of host-based iWARP is expected since it performs all tasks, including CRC, in software. However, surprisingly the CPU utilization of host-assisted iWARP is even higher than host-based iWARP. This is attributed to the higher performance of host-assisted iWARP. Note that host-assisted iWARP performs a copy of the data into a temporary buffer (to insert markers) before transmitting. Since the performance of host-assisted iWARP is higher than that of host-based iWARP, the data is sent out of the temporary buffer faster, thus requiring the CPU to spend a larger fraction of the time performing memory copy.

### 4.2.2 Impact of Out-of-Cache Communication

Figure 11 shows a comparison of the bandwidths achieved by the different iWARP designs when the data being sent is not in the system cache. This experiment is similar to the bandwidth test described in section 4.2.1, the only difference being that the sender sends out a number of *different* messages, each of size *S*, instead of sending the *same* message multiple times. As illustrated in the figure, we see a reversal in the relative performances of host-assisted and host-offloaded iWARP as compared to the regular bandwidth test (Figure 10(b)). This trend is associated to the high dependency of the performance of copy-based schemes such as host-based and host-assisted iWARP on cache activity. In other words, since host-based and host-assisted iWARP perform a copy of the data being communicated, when not in cache this data has to be fetched from memory, significantly altering their performance.

To further analyze out-of-cache communication, we show the ratio of cache traffic to the number of bytes communicated in Figure 12 (transmit and receive), i.e., if for communicating $N$ bytes of data, a cache traffic of $M$ bytes is generated, the ratio is represented as $M / N$. The cache traffic is measured based on the Pentium hardware performance measurement counters (PMCs) provided by the processor subsystem. In these figures, three important things are to be noticed:

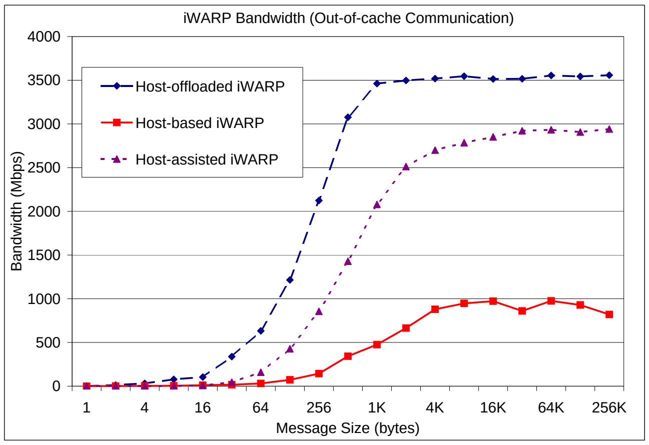


Figure 11: Out-of-Cache Communication

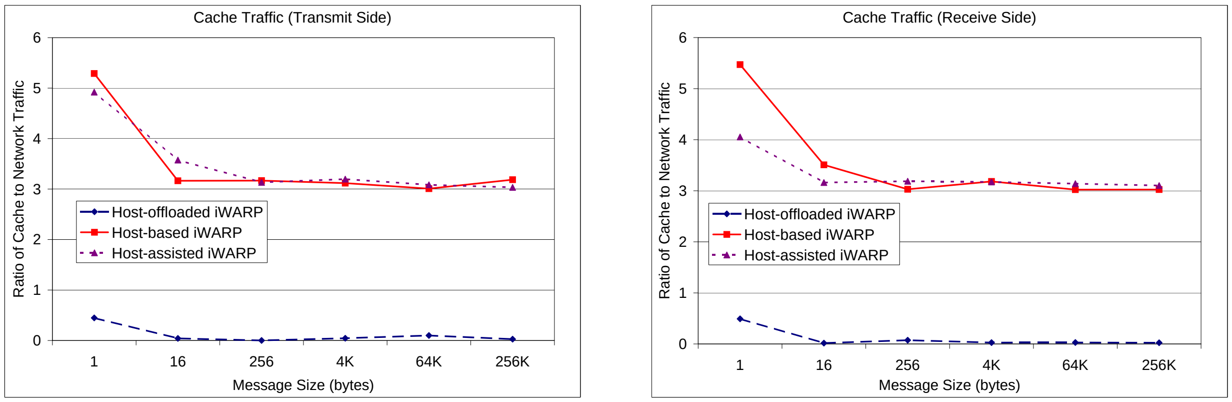
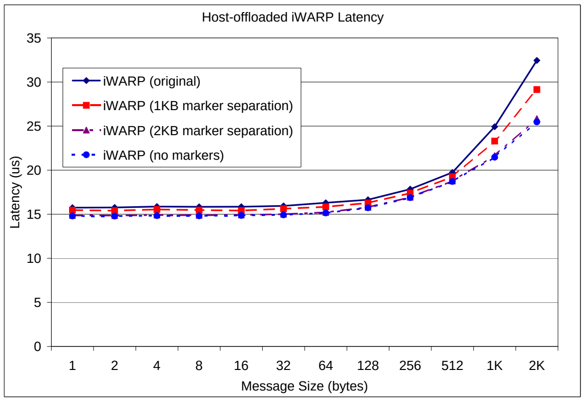


Figure 12: iWARP Out-of-Cache Communication Bandwidth: (a) Transmit Side, (b) Receive Side

1. There is a huge difference in the cache traffic generated between host-offloaded iWARP and the copy-based schemes (host-assisted and host-based iWARP). In fact, the cache traffic in host-offloaded iWARP is very close to zero. This result is not surprising since in the host-offloaded iWARP design, the CPU does not touch the data that is being transmitted or received. Thus, it does not generate much cache traffic irrespective of whether the data to be communicated is already in cache or not.



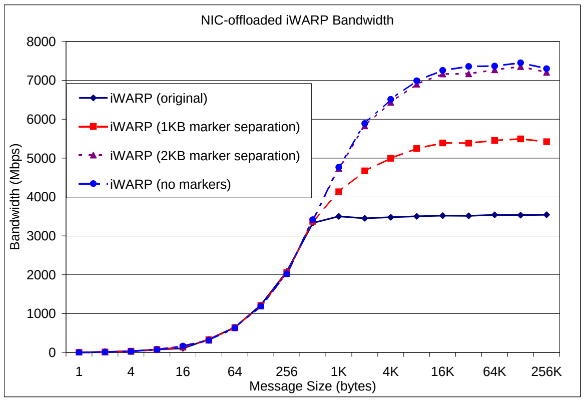


Figure 13: Impact of marker separation on iWARP performance: (a) Latency (b) Bandwidth

2. Both the copy-based designs of iWARP generate about 3 bytes of cache traffic for each byte transferred over the network. This trend closely matches previous literature [16, 20]. Specifically, if a copy needs to be performed in the x86 architecture, both the source and destination buffers need to be in cache. Further, if the receive buffer is larger than what the cache can accommodate, flushing it to memory would generate cache traffic as well. In summary, for a copy when the source and destination buffers are neither cached nor small enough to be accommodated in it, for each byte to be transferred two bytes have to fetched into the cache and one byte has to be flushed out – a total of 3 bytes. Our experimental results match this trend for all, except very small message sizes.

3. For very small messages, we notice that the cache traffic is much higher than three times the number of bytes transmitted. This is attributed to the system noise that is caused by other cache misses, e.g., control structures of the communication stack. For larger messages, however, the overhead caused by such noise is negligible and thus cannot be noticed in the graph.

#### 4.2.3 Impact of Marker Separation Length

The iWARP standard specifies a length of 512 bytes between the markers in order to ensure that each Ethernet packet has at least one marker in it. Increasing this length can result in some packets not containing markers. Thus, if such a packet arrives out of order, it has to be either dropped or buffered on the NIC. In this section, we vary the separation length between the markers in the MPA protocol and study its impact on the performance of the different iWARP designs.

In our experiments, we noticed that the performance of host-assisted and host-based iWARP does not vary much with marker separation length. This is expected, since these designs do not perform any task which can be affected by it. The amount of data copied or transferred varies, but minimally. For host-offloaded iWARP, however, the marker separation length makes a big difference, i.e., it directly impacts the number of DMA operations needed, e.g., for a 16KB message, with a marker separation length of 512 bytes about 64 DMAs are needed, while with a marker separation length of 1KB only 32 are needed. Figure 13 shows the impact of varying the marker separation on the performance of host-offloaded iWARP. We use the latency/bandwidth tests described in the section above, but vary the marker separation from 512 bytes (as per the iWARP standard) to infinite bytes (no markers). We observe that for larger marker sizes, the performance of the host-offloaded design improves to about 7.2Gbps, which is higher than what host-assisted iWARP can achieve (Figure 10(b)).

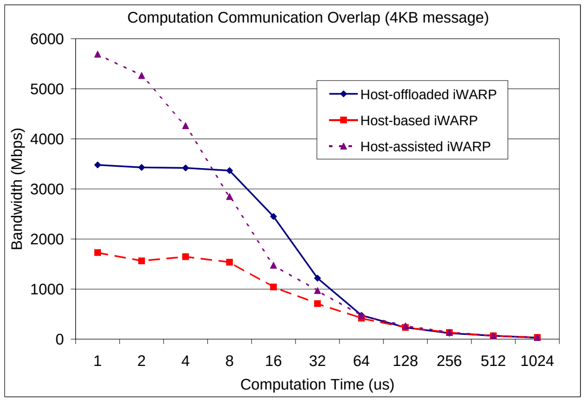
The iWARP (no markers) case depicts the behavior of current iWARP adapters which only support in-order packets and drop any out-of-order packet received. The iWARP (512 byte marker separation) case depicts the behavior of feature-complete iWARP implementations that strictly follow the iWARP standard. All other implementations are intermediate cases which we trade NIC buffering capability for in-order communication performance. Though these cases are not a part of the current iWARP standard, we still study them as they have the potential to be included in future revisions of the standard based on their impact on the performance of iWARP.

#### 4.2.4 Computation and Communication Overlap Capability

Figure 14 shows the capabilities of the iWARP designs to overlap computation with communication. This experiment is similar to the bandwidth test, except that before each request is initiated, interleaving computation is added. Bandwidth is measured for different amounts of computation. As shown in the figure, for very little computation, the trend is similar to the bandwidth test (Figure 10(b)), with host-assisted iWARP performing the best, followed by host-offloaded and host-based iWARP, respectively. However, as the computation increases, the performance of host-assisted iWARP drops rapidly. When the amount of computation becomes higher than 8$\mu$s for 4KB messages and 128$\mu$s for 128KB messages, host-offloaded iWARP outperforms host-assisted iWARP. This behavior is attributed to the large amount of CPU used by host-assisted iWARP as compared to host-offloaded iWARP. Due to this, any additional computation added leads to lesser CPU cycles allotted to communication, resulting in performance loss. On the other hand, in host-offloaded iWARP, since the NIC is performing most of the communication tasks, additional computation on the CPU does not affect it as much.

### 4.3 Iso-surface Visual Rendering Application

Iso-surface rendering [17, 3, 10] is widely used technique used in many application areas including environmental simulations, biomedical images and oil reservoir simulators, for extracting and visualizing surfaces within a 3D



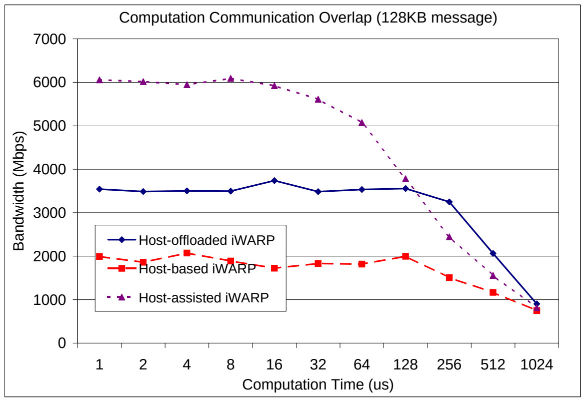
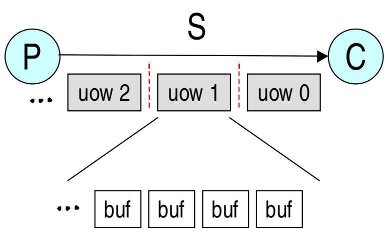


Figure 14: Computation and communication overlap capability of the different iWARP implementations: (a) Message size 4KB, (b) Message Size 128KB

volume. In this paper, we utilize a component-based implementation of iso-surface rendering [7], developed by University of Maryland, on top of their widely used data-cutter library [6]. The original implementation of data-cutter is based on TCP/IP sockets. In this paper, we modified the components of data-cutter that are relevant to the iso-surface application, to directly communicate using the native iWARP interface. This allows us to evaluate this application using the different iWARP designs proposed in this paper.

**Overview of the Data-cutter Library:** Data-cutter is a component framework that supports subsetting and user-defined processing of large multi-dimensional datasets. It provides a framework, called filter-stream programming, for developing data-intensive applications. In this framework, the application processing structure is implemented as a set of components, called *filters*. Data exchange between filters is performed through a *stream* abstraction. A *stream* denotes a unidirectional data flow from one filter (i.e., the producer) to another (i.e., the consumer). The overall processing structure of an application is realized by a *filter group*, which is a set of filters connected through logical streams. When a filter group is instantiated to process an application query, the run-time system establishes connections between filters placed on different hosts before starting the execution of the application query. Filters placed on the same host execute as separate threads. An application query is handled as a *unit of work* (UOW) by the filter group. The size of the UOW also represents the granularity in which data segments are distributed in the system. The processing of a UOW can be done in a pipelined fashion; different filters can work on different data elements simultaneously, as shown in Figure 15. Several data-intensive applications including the iso-surface visual rendering application, have been designed and developed using the data-cutter run-time framework.

**Evaluating the Iso-surface Application:** Figure 16 shows the execution time for the iso-surface application



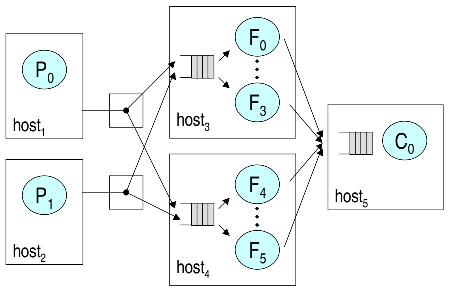
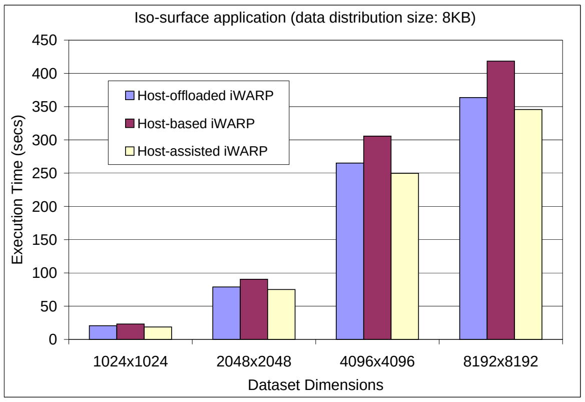


Figure 15: Data-Cutter stream abstraction and support for copies. (a) Data buffers and end-of-work markers on a stream. (b) P,F,C filter group instantiated using transparent copies.



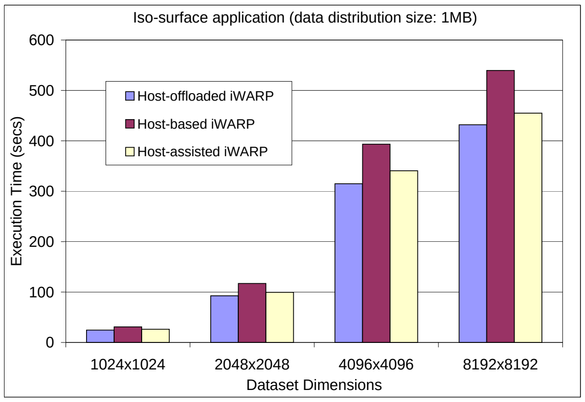


Figure 16: Iso-surface visual rendering application: (a) UOW 8KB, (b) UOW 1MB

using the different iWARP designs for two granularities of data distribution. Figure 16(a) uses a data distribution granularity (UOW) of 8KB, while Figure 16(b) uses a granularity of 1MB. The complete dataset is about 1GB in size, which is hosted on a *ram disk* in order to avoid disk fetch overheads in the experiment. The application uses four filters, *read dataset*, *isosurface extraction*, *shade and rasterize* and merge/view. As mentioned earlier, each filter performs some computation and communicates the processed data to the next filter. Once the communication is initiated, the filter starts computation on the next UOW, thus attempting to overlap communication with computation. In our experiments, two instances of the four filters (i.e., totally eight filters) were placed on the four dual-processor nodes.

As shown in Figure 16(a), when the UOW is small (8KB), host-assisted iWARP performs the best, followed by host-offloaded iWARP. Host-based iWARP performs the worst of the three. These results are consistent with the raw latency and bandwidth performance (Figure 10). Though the application tries to overlap computation and communication, the amount of computation is quite small (since the UOW is small); thus the lesser CPU utilization of host-offloaded iWARP does not help much in this case. With a larger UOW (1MB), we notice a reversal in trend with host-offloaded iWARP outperforms host-assisted iWARP. This is attributed to the cache misses associated with this workload as well as the lower CPU usage of host-offloaded iWARP. The experimental systems we used support only 512KB of L2 cache. Thus, with a UOW of 1MB, data processing has to deal with cache misses. Further, for host-assisted iWARP, even the communication has to deal with cache misses, because of the copies associated with it.

When the UOW is small (e.g., 8KB), the data that is fetched to cache for computation can be reused for performing the copy associated with communication; thus there are no additional cache misses. However, when the UOW is large (e.g., 1MB), the data has to be fetched to cache in parts, and flushed back in order to accommodate later parts of the data. Thus, during communication, the data has to fetched to cache again, resulting in more cache misses. On the other hand, host-offloaded iWARP does not have to deal with the extra cache misses during communication because of its zero-copy capability, which helps it achieve better performance.

## 5 Related Work

There has been a lot of prior research on 10GE [18, 15] as well as iWARP [4, 13, 21]. However, these prior designs do not study the impact of out-of-order arrival of packets in iWARP. Some of them do not utilize the hardware capabilities of network adapters either. Duato et. al. [24, 23], have performed previous research on dealing with multi-path communication for IB. Their solution utilizes limiting the number of packets injected in the network and utilizing buffering capabilities on the NIC on the receiver end to reorder out-of-order packets. Though this work has only been performed with respect to IB, we believe that it strongly complements our work, i.e., once the iWARP frames are identified and extracted from out-of-order TCP/IP packets, the schemes proposed by Duato can be utilized to efficiently reorder them on the NIC. In summary, this paper significantly differs, yet complements strongly with prior research and thus makes a novel and interesting contribution.

## 6 Concluding Remarks

Multi-path communication is gaining significant prominence with the growing scales of high-end computing (HEC) systems and the increasing focus on capabilities to tolerate hardware faults and congestion. However, with multi-path networks the communication protocols face the disadvantage of having to deal with out-of-order arrival of packets. Internet Wide-Area RDMA Protocol (iWARP) is a new initiative as a high-speed communication protocol over 10GE, with the unique ability to maintain backward compatibility with the existing TCP/IP infrastructure. This, however, makes it more complicated and expensive to correctly identify and understand the contents of iWARP packets during out-of-order communication. More importantly, it makes the packet format and processing significantly more complex, which unfortunately affects the performance of in-order communication as well. In this paper, we analyzed the trade-offs in designing a feature-complete iWARP stack, i.e., one which provides support for out-of-order arriving packets and thus multi-path systems, while focusing on the performance of in-order communication. Specifically, we proposed three different feature-complete designs of iWARP: (i) host-based iWARP, (ii) host-offloaded iWARP, and (iii) host-assisted iWARP. We analyzed each of these designs using micro-benchmarks as well as an iso-surface visual rendering application and demonstrated that depending on the characteristics of the upper-layers or applications, different iWARP designs can provide the best overall performance.

## References

[1] InfiniBand Trade Association. http://www.infinibandta.com.

[2] S. Bailey and T. Talpey. Remote Direct Data Placement (RDDP), April 2005.

[3] C. L. Bajaj, V. Pascucci, D. Thompson, and X. Y. Zhang. Parallel Accelerated Isocontouring for Out-of-core Visualization. In *Proceedings of the IEEE Symposium on Parallel Visualization and Graphics*, pages 97–104, San Francisco, CA, Oct 1999.

[4] P. Balaji, H. W. Jin, K. Vaidyanathan, and D. K. Panda. Supporting iWARP Compatibility and Features for Regular Network Adapters. In *RAIT*, 2005.

[5] P. Balaji, H. V. Shah, and D. K. Panda. Sockets vs RDMA Interface over 10-Gigabit Networks: An In-depth analysis of the Memory Traffic Bottleneck. In *RAIT*, 2004.

[6] M. D. Beynon, T. Kurc, U. Catalyurek, C. Chang, A. Sussman, and J. Saltz. Distributed Processing of Very Large Datasets with DataCutter. *Parallel Computing*, October 2001.

[7] M. D. Beynon, T. Kurc, U. Catalyurek, and J. Saltz. A Component-based Implementation of Iso-surface Rendering for Visualizing Large Datasets. *Report CS-TR-4249 and UMIACS-TR-2001-34, University of Maryland, Department of Computer Science and UMIACS*, 2001.

[8] R. V. Boppana and S. Chalasani. Fault-Tolerant Wormhole Routing Algorithms for Mesh Networks. *IEEE Transactions on Computers*, pages 848–864, July 1995.

[9] S. Herrmann M. Castagnoli, G. Brauer. Optimization of cyclic redundancy-check codes with 24 and 32 paritybits. In *IEEE Transactions on Communication*, 1993.

[10] Y. J. Chiang and C. Silva. External Memory Techniques for Isosurface Extraction in Scientific Visualization. In *External Memory Algorithms and Visualization*, volume 50, pages 247–277, 1999.

[11] A. Cohen, S. Rangarajan, and H. Slye. On the Performance of TCP Splicing for URL-aware Redirection. In *USENIX '99*.

[12] P. Culley, U. Elzur, R. Recio, and S. Bailey. Marker PDU Aligned Framing for TCP Specification, November 2002.

[13] D. Dalessandro, A. Devulapalli, and P. Wyckoff. Design and Implementation of the iWARP Protocol in Software. In *PDCS '05*.

[14] W. Feng, P. Balaji, C. Baron, L. N. Bhuyan, and D. K. Panda. Performance Characterization of a 10-Gigabit Ethernet TOE. In *HotI*, 2005.

[15] W. Feng, J. Hurwitz, H. Newman, S. Ravot, L. Cottrell, O. Martin, F. Coccetti, C. Jin, D. Wei, and S. Low. Optimizing 10-Gigabit Ethernet for Networks of Workstations, Clusters and Grids: A Case Study. In *SC '03*.

[16] A. Foong, H. Hum, T. Huff, J. Patwardhan, and G. Regnier. TCP/IP Performance Revisited. In *ISPASS*, 2003.

[17] J. Gao and H. Shen. Parallel view dependent isosurface extraction using multi-pass occlusion culling. In *Proceedings ACM/IEEE Symposium on Parallel and Large Data Visualization and Graphics*. ACM SIGGRAPH, 2001.

[18] G. Hurwitz and W. Feng. Initial End-to-End Performance Evaluation of 10-Gigabit Ethernet. In *HotI*, 2003.

[19] NetEffect Inc. http://www.neteffect.com/product-features.html.

[20] H. W. Jin, P. Balaji, C. Yoo, J . Y. Choi, and D. K. Panda. Exploiting NIC Architectural Support for Enhancing IP based Protocols on High Performance Networks. *JPDC '05*.

[21] H. W. Jin, S. Narravula, G. Brown, K. Vaidyanathan, P. Balaji, and D. K. Panda. Performance Evaluation of RDMA over IP: A Case Study with the Ammasso Gigabit Ethernet NIC. In *HPI-DC*, 2005.

[22] H. M. Khosravi and A. Foong. Performance Analysis of iSCSI and Effect of CRC Computation. In *BEACON '04*.

[23] M. Koibuchi, J. C. Martinez, J. Flich, A. Robles, P. Lopez, and J. Duato. Enforcing In-Order Packet Delivery in PC Clusters using Adaptive Routing. *Journal of Parallel and Distributed Processing*, 2007.

[24] J. C. Martinez, J. Flich, A. Robles, P. Lopez, J. Duato, and M. Koibuchi. In-Order Packet Delivery in Interconnection Networks using Adaptive Routing. In *IPDPS*, 2005.

[25] T. Otsuka, M. Koibuchi, T. Kudoh, and H. Amano. Switch-tagged VLAN Routing Methodology for PC Clusters with Ethernet. In *the Proceedings of the IEEE International Conference on Parallel Processing (ICPP)*, 2006.

[26] F. Petrini, W. C. Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The Quadrics Network (QsNet): High-Performance Clustering Technology. In *Hot Interconnects*, 2001.

[27] G. F. Pfister and V. A. Norton. Hot-spot Contention and Combining in Multistage Interconnection Networks. *IEEE Transactions on Computers*, 34:943–948, 1985.

[28] D. V. Sarvate. Computation of cyclic redundancy checks via table look-up. In *Communications of the ACM*, volume 31, 1998.

[29] IEEE 802 Standards. IEEE 802.1Q – Virtual LANs.

[30] J. Stone and C. Partridge. When the CRC and TCP Checksum Disagree. In *ACM SIGCOMM*, 2000.

[31] A. Vishnu, M. Koop, A. Moody, A. Mamidala, S. Narravula, and D. K. Panda. Hot-Spot Avoidance With Multi-Pathing Over InfiniBand: An MPI Perspective. In *CCGrid*, 2007.

[32] A. Vishnu, A. R. Mamidala, and D. K. Panda. Performance Modeling of Subnet Management on Fat Tree InfiniBand Networks using OpenSM. In *Workshop on System Management Tools on Large Scale Parallel Systems*, 2005.