EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing

Piyush Shivam
Computer/Information Science
The Ohio State University
2015 Neil Avenue
Columbus, OH 43210
shivam@cis.ohio-state.edu

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

Dhabaleswar Panda
Computer/Information Science
The Ohio State University
2015 Neil Avenue
Columbus, OH 43210
panda@cis.ohio-state.edu

ABSTRACT

Modern interconnects like Myrinet and Gigabit Ethernet offer Gb/s speeds which has put the onus of reducing the communication latency on messaging software. This has led to the development of OS bypass protocols which removed the kernel from the critical path and hence reduced the end-to-end latency. With the advent of programmable NICs, many aspects of protocol processing can be offloaded from user space to the NIC leaving the host processor to dedicate more cycles to the application. Many host-offload messaging systems exist for Myrinet; however, nothing similar exits for Gigabit Ethernet. In this paper we propose Ethernet Message Passing (EMP), a completely new zero-copy, OS-bypass messaging layer for Gigabit Ethernet on Alteon NICs where the entire protocol processing is done at the NIC. This messaging system delivers very good performance (latency of 23 us, and throughput of 880 Mb/s). To the best of our knowledge, this is the first NIC-level implementation of a zero-copy message passing layer for Gigabit Ethernet.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Protocol Architecture*

General Terms

Design, Performance

Keywords

Gigabit ethernet, message passing, OS bypass, user level protocol

1. INTRODUCTION

High-Performance computing on a cluster of workstations requires that the communication latency be as small as possible. The communication latency is primarily composed of two components: time spent in processing the message and the network latency (time on wire). Modern high speed interconnects such as Myrinet and Gigabit Ethernet have shifted the bottleneck in communication from the interconnect to the messaging software at the sending and receiving ends. In older systems, the processing of the messages by the kernel caused multiple copies and many context switches which increased the overall end-to-end latency. This led to the development of user level network protocols where the kernel was removed from the critical path of the message. This meant that the parts of the protocol or the entire protocol moved to the user space from the kernel space. One of the first examples in this case is U-Net [18].

Another development which took place was the improvement of the network interface card (NIC) technology. In the traditional architecture, the NIC would simply take the data from the host and put it on the interconnect. However, modern NICs have programmable processors and memory which makes them capable of sharing some of the message processing work with the host. Thus, the host can give more of its cycles to the application, enhancing application speedup. Under these two developments, modern messaging systems are implemented outside the kernel and try to make use of available NIC processing power.

Since its inception in the 1970s, Ethernet has been an important networking protocol, with the highest installation base. Gigabit Ethernet builds on top of Ethernet but increases speed multiple times (Gb/s). Since this technology is fully compatible with Ethernet, one can use the existing Ethernet infrastructure to build gigabit per second networks. A Gigabit Ethernet infrastructure has the potential to be relatively inexpensive, assuming the industry continues ethernet price trends and completes the transition from fiber to copper cabling.

Given so many benefits of Gigabit Ethernet it is imperative that there exist a low latency messaging system for Gigabit Ethernet just like GM [2], FM [13], AM2 [5], and others for Myrinet. However, no NIC-level, OS-bypass, Zero-copy messaging software exists on Gigabit Ethernet. All the efforts until now like the GAMMA [4], MESH [3], MVIA [11], GigaE PM [16], and Bobnet [6] do not use the capabilities of the programmable NIC to offload protocol processing onto

SC2001 November 2001, Denver, CO, USA

Table 1: Classification of existing message passing systems.

	Programmable NIC	Non Programmable NIC
Myrinet	**GM, FM, AM2**	(n/a)
Gigabit Ethernet	**EMP** (This Work)	**MVIA, GigaE-PM, Bobnet MESH, GAMMA**

the NIC. Table 1 shows the classification of existing message passing systems along with EMP.

These Ethernet messaging systems use non-programmable NICs. Thus, no more processing can be offloaded onto the NIC than which has been already been burned into the card at design time. There was work done in the area of message fragmentation where the NIC firmware was modified to advertise a bigger MTU to IP which was then fragmented at the NIC [7]. The Arsenic project [15] extends a packet send and receive interface from the NIC to the user application. And a port of the ST protocol has been written which extends the NIC program to bypass the operating system [14]. However, these are not complete reliable NIC-based messaging systems. As indicated in [10] it is desirable to have most of the messaging overhead in the NIC, so that the processing power at the host can be used for applications.

In this paper we take on a challenge of designing, developing and implementing a zero-copy, OS-bypass, NIC-level messaging system for Gigabit Ethernet with Alteon NICs. OS bypass means that the OS is removed from the critical path of the message. Data can be sent/received directly from/into the application without intermediate buffering making it a true zero-copy architecture. The protocol has support for retransmission and flow control and hence is reliable. All parts of the messaging system are implemented on the NIC, including message-based descriptor management, packetization, and reliability. In fact, the host machine does not interfere with a transfer after ordering the NIC to start it, thus the adjective NIC-driven.

Section 2 of this paper provides an overview of the Alteon NIC. Section 3 describes the design challenges faced while developing EMP. In Section 4 we give the implementation details. Finally we provide the results of our experiments in Section 5, which show small message latencies around 20 us, and large message bandwidths around 880 Mb/s.

2. ALTEON NIC OVERVIEW

Alteon Web Systems, now owned by Nortel Networks, produced a Gigabit Ethernet network interface chipset based around a general purpose embedded microprocessor design which they called the Tigon2. It is novel because most Ethernet chipsets are fixed designs, using a standard descriptor-based host communication protocol. A fully programmable microprocessor design allows for much flexibility in the design of a communication system. This chipset was sold on boards by Alteon, and also was used in board designs by other companies, including Netgear. Broadcom has a chip (5700) which implements the follow-on technology, Tigon3, which should be similar enough to allow use of our messaging environment on future gigabit ethernet hardware.

The Tigon chip is a 388-pin ASIC consisting of two MIPS-like microprocessors running at 88 MHz, an internal memory bus with interface to external SRAM, a 64-bit, 66 MHz PCI interface, and an interface to an external MAC. The chip also includes an instruction and data cache, and a small amount of fast per-CPU "scratchpad" memory. The instruction set used by the Tigon processors is essentially MIPS level 2 (as in the R4000), without some instructions which would go unused in a NIC application. Hardware registers can be used by the processor cores to control the operation of other systems on the Tigon, including the PCI interface, a timer, two host DMA engines, transmit and receive MAC FIFOs, and a DMA assist engine. Our particular cards have 512 kB of external SRAM, although implementations with more memory are available. The NIC exports a 4 kB PCI address space, part of which can be positioned by the host to map into any section of the full 512 kB external SRAM.

Although we do not use it, Alteon freely distributes the source code for their driver and firmware which mimics a standard fixed descriptor host interface model suitable for kernel-level IP networking. The rest of this paper details how we replace the entire standard driver and firmware with our own novel protocol designed specifically for message passing.

3. DESIGN CHALLENGES

Our goal was to develop a design for a new firmware for the Tigon, and associated code running in the host, to facilitate OS-bypass high-throughput and low-latency message passing communications, with no buffer copies and no intervention by the operating system. We reuse none of the original firmware written by Alteon, and know of no other original firmware designs, although quite a number of projects have made small modifications to the original Alteon firmware.

The following sections highlight some of the major challenges we faced in designing both the network protocol and the interface to a user application through MPI. Figure 1 shows the core components addressed in the following. The existing solution in the figure refers to the current messaging layer implementations on the Gigabit Ethernet.

3.1 Protocol Processing

One of the most important design challenges was to decide how to distribute the various functions related to protocol processing such as reliability, fragmentation, *etc.*: on the host, NIC, or shared in both? In EMP we have offloaded all the protocol processing to the NIC because of its unique processing capabilities (two 88 MHz MIPS processors). That is the reason for calling EMP NIC-driven.

3.1.1 Framing

Once the NIC has been informed that there is data to send, it retrieves the data from the host in preparation for moving the data to the network. Here, we had to decide whether we should buffer the data at the NIC while sending and receiving the message. We decided against it because buffering

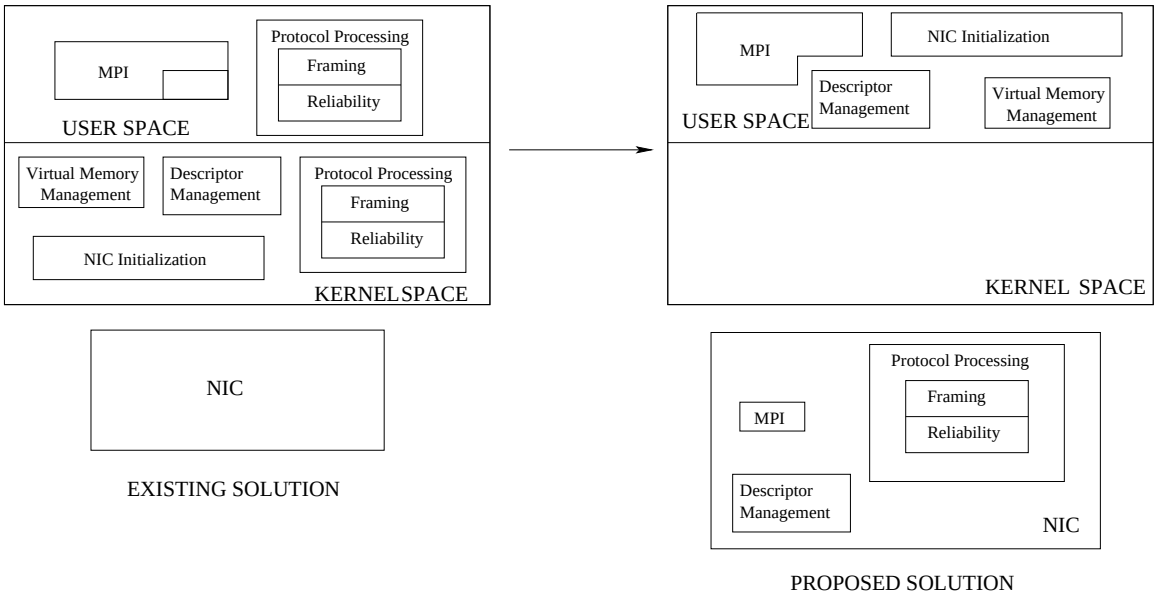

Figure 1: Our approach in the redistribution of the various components of the messaging system.

would add greatly to the system overhead. Instead, whenever the NIC has data to send, it pulls the data from the host one frame at a time and puts it on the wire. Similarly on the receiving side, if there is no pre-posted receive for the incoming frame, the frame is simply dropped to avoid buffering. MPI programs frequently arrange to pre-post all receives to avoid buffering by the messaging system, because users know that it hurts performance and that the buffering limit varies among implementations, thus our design caters to this type of usage. VIA [17] also imposes this condition in its design.

3.1.2 Reliability

EMP presents a reliable transport interface to the host. While designing the reliability of EMP, one of the design issues [1] we faced was the unit of acknowledgement (the number of frames to acknowledge at one time). We decided to acknowledge a collection of frames instead of a single frame (each message is made up of multiple frames). There were two main reasons for that. The Ethernet frame size is small so acknowledging every frame would introduce a lot of overhead. Moreover, since we were developing our messaging layer for LAN clusters where the rate of frame loss is not as excessive as WAN, we could decide not to acknowledge every frame. EMP has the flexibility of configuring the unit of acknowledgement on the fly depending upon application and network configuration. This feature also allows us to research reliability issues in the future for different kinds of applications.

3.2 Virtual memory management

There are two mechanisms available by which data can be moved to the NIC to allow it to be sent to the network, or the reverse, as defined by the PCI specification. The first is programmed input/output (PIO), where the host CPU explicitly issues a write instruction for each word in the message to be transferred. The second is direct memory access (DMA), where the NIC can move the data directly from the host memory without requiring the involvement of the host CPU. We quickly decided that PIO would require extremely much host overhead, and thus use DMA for the movement of all message data.

However, this decision leads us to solve another problem. All widely used operating systems employ virtual memory as a mechanism of allowing applications to exceed physical memory constraints, and to implement shared libraries, and to provide a wide array of other handy features. Although an application on the host machine does not need to know about the actual physical layout of its data, the NIC is required to perform all its DMA transfers to and from the host using physical addresses. Some entity must translate from user virtual to PCI physical addresses for every page in the transfer.

One option would be to have the NIC maintain a copy of the page tables which the kernel and processor use to establish the mapping. If we devoted all of the memory on the NIC to page tables, we could map no more than 512 MB of process address space on the x86 architecture. This may be sufficient for some applications, but was discarded as being too constraining. We considered caching just the "active" parts of the page tables, meaning only those pages which are being used in send/receive messages are available on the NIC, again with some sort of kernel interaction. Kernel virtual memory operation overhead, complexity, and again limited NIC memory all suggest that even a caching version would not perform well.

This leads us to mandate that the user application specifically inform the NIC of the physical address of each page involved in the transfer. However, user space does not have access to this information which resides in the kernel. In the implementation section below we point out that it does not cost too much to use a system call to ask the kernel for the translations.

Applications which use the NIC with our firmware must ensure that the physical pages comprising a message, either for transmit or receive, stay resident at all times during their potential use by the NIC. Unix provides a system call, `mlock`, which selectively guarantees this for individual pages, or the variant, `mlockall`, which can be used to inform the operating system that all parts of the address space should remain in physical memory. We choose the latter method which requires only one system call during initialization and is appropriate for most memory-resident codes, but that technique precludes the use of swap space. The alternative of individual page mapping is potentially more expensive depending on the exact distribution of memory used for communication with the network, although the per-page mapping cost is less than a microsecond on our hardware.

3.3 Descriptor management

We use the term "descriptor" to refer to the pieces of information that describe message transfers, and which must pass between the host system and the NIC to initiate, check, and finalize sends and receives. Descriptors are usually small in comparison to the message payloads they represent.

To initiate a send, or to post a receive, the host must give to the NIC the following information: location and length of the message in the host address space, destination or source node, and the MPI-specified message tag.

This information is passed to the NIC by "posting" a descriptor. The NIC needs access to descriptor fields, which are generated in the host. It can get this information either by individually DMAing fields on demand from the host or it can keep a local copy of the descriptor in NIC memory. The tradeoff is between expensive DMA operations (1–3 us per transfer) and the number of descriptors which can reside in the limited NIC memory. Too many accesses to descriptor fields via DMA will degrade performance immensely.

We found that the NIC indeed needs to access the fields of a descriptor at many stages during the lifetime of a send or receive message, thus it requires a copy of this descriptor locally and can not use DMA on demand to retrieve it with any reasonable performance. The host, on the other hand, fills descriptors then does not need to be concerned with them until message completion time, *i.e.,* at the conclusion of a possibly asynchronous operation.

Since we require a local copy on the NIC, and since the host will not frequently use the descriptor, we chose to allocate permanent storage for an array of descriptors in the NIC memory. The host accesses these using PIO, which is slow in bulk compared to DMA, but provides lower latency for small transfers. Details of our extended descriptor formats for transfers greater than a page are provided in the implementation sections below.

The one piece of information that the host needs from the NIC after an asynchronous send or receive has been initiated is, "Is it done yet?" Traditional Ethernet devices communicate this by an interrupt to the operating system kernel, which is clearly inappropriate given our low latency goals. Other systems (*e.*g., VIA, Infiniband [8]) use a completion queue, which is a list describing the transfers that have completed. Our project goals state that no buffering will ever be performed, which implies that no unexpected messages will ever be received at any node, which reduces the concept of a completion queue to simply reporting statuses of transfers that the host explicitly initiated. Since the host has this knowledge, it can instead remember which descriptors it has posted, and explicitly poll them when it is interested in determining if the transfer has completed.

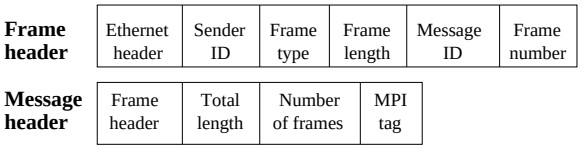

Figure 2: Two main headers used in EMP.

4. IMPLEMENTATION

Our firmware consists of 12,000 lines of C code, and another 100 lines of MIPS assembly. Roughly half of the code deals with the basic hardware functionality of the various Tigon subsystems, including DMA engines, MAC interface, link negotiation and control. The core of the code is an event loop written in three assembly instructions which tests for the next pending event and dispatches control to the appropriate handling routine. Details of our implementation of fundamental modules are provided in the following sections.

4.1 Protocol Processing

A message is composed of multiple frames. A message is a user defined entity whereas the frame is defined by the hardware. The length of each frame is 1500 bytes, but could be larger if the particular Ethernet switch infrastructure provides support for 9000 byte "jumbo" frames. The first frame of the message contains a message header which embeds frame header. All frames contain a frame header, and the first frame of a message also contains extra per-message information as shown in Figure 2.

For each host we keep a small amount of information including expected next message ID, but we have no concept of a connection, and thus expect not to be limited in scalability by that. Other protocols generally arbitrate a unique connection between each pair of communicating hosts which preserves the relatively large state for that pairwise conversation. The per-host state we keep is on each NIC in the cluster is: remote MAC address, send and receive sequence numbers (message ID).

For each message we keep a data structure describing the state of the transfer for just that message, which includes a pointer to the descriptor filled by the host. The limits to scalability of the system thus mirror the scalability limits inherent in each application, as represented by the number of outstanding messages it demands. Each of these message states is much larger than the per-host state mentioned above.

Once the NIC is informed that a message is there to send it DMAs the data from the application space and prepends the message/frame header before sending it on the wire. As mentioned before we have something called a collection

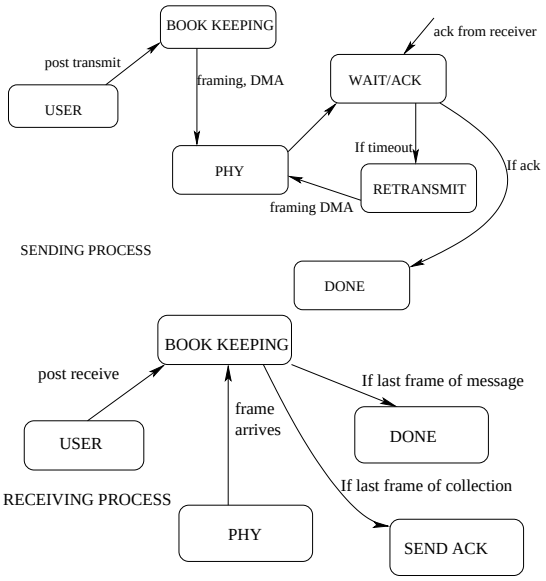

Figure 3: Processing flow for sending and receiving.

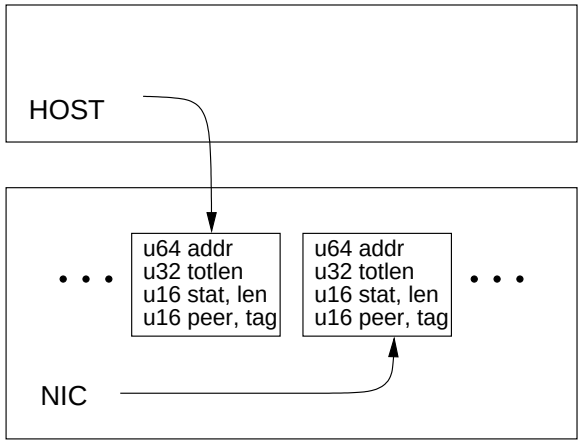

Figure 4: Some short-format descriptors in the NIC memory, showing ownership by either the HOST or the NIC.

of frames. In the current implementation this number is 3. Thus, once 3 consecutive frames have been sent the timer for that collection starts. Any time the receiver receives all the frames in that collection, the acknowledgement for that collection is sent. The acknowledgement consists of the frame header of one of the frames for that collection. Essentially it is the header of the last received frame of that collection. If the sender receives an acknowledgement it verifies whether all the frame collections have been acknowledged. If yes, then the host is informed that the message has been sent and it is removed from consideration by the NIC. If any collection times out then all the frames of that collection are resent. We have provided support for out of order frames too in EMP.

The sequence of operations involved in sending/receiving are shown in Figure 3. The USER (application space) posts a transmit or a receive. The data structures corresponding to that operation are updated (BOOKKEEPING). In case of transmit a DMA is done and the message is sent to the wire (PHY) as explained above. A timer corresponding to that transmit starts and a RETRANSMIT is performed if that timeout value is reached before the remote ACK arrives. Similarly, on the receive side when the frame arrives, a check is made if it is the last frame (BOOKKEEPING), in which case the receive is marked DONE and an acknowledgement is sent (SEND ACK).

4.2 Virtual memory and descriptors

As discussed in the design section, we translate addresses from virtual to physical on the host. This is done by the kernel and invoked from the user application (library) as an `ioctl` system call. This overhead is included in the performance results below.

The descriptors which are used to pass the information between the host and the NIC each require 20 bytes of memory, and we currently allocate static storage on the NIC for 640 of these to serve both message receives and sends. The physical address of every page in a transfer must be passed to the NIC, which leads us to define multiple descriptor formats to include these addresses.

The first simply includes a single 8-byte address in the 20-byte descriptor and can describe a message up to a page in size. The second is built by chaining up to four descriptors which among them describe a message up to seven pages long (28 kB on x86). The descriptor format used for messages larger than this is similar to the short format in that only one address is given to the NIC, but this address is a pointer to a page full of addresses which the NIC uses indirectly to get the address of any single page in the transfer. In all these formats we always provide a 16-bit length field along with each page address which allows us to do scatter/gather transfers, so that if an application specifies a message which is not even contiguous in user memory (such as with an arbitrary MPI Datatype), the NIC can gather the individual chunks when preparing a message for the network, and the reverse when receiving a network frame.

A diagram of the descriptor layout is shown in Figure 4, where all the descriptors physically reside in NIC memory, but the host will take ownership of some when it needs to post a new send or receive. After writing the relevant information for the transfer, ownership is passed to the NIC (by modifying the "stat" field), which will then carry out the transfer and return ownership to the host.

5. PERFORMANCE EVALUATION

We compare basic networking performance with two other systems which are frequently used for MPI message passing. The base case for using Gigabit Ethernet is to use standard TCP/IP, which uses exactly the same Alteon NIC, but running the default firmware coupled with a TCP stack in the kernel. The second system is GM over Myrinet, which uses different hardware, but shares with EMP the NIC programmability features and the focus on message passing.

5.1 Experimental Setup

For the Gigabit Ethernet tests, we used two dual 933 MHz Intel PIII systems, built around the ServerWorks LE chipset

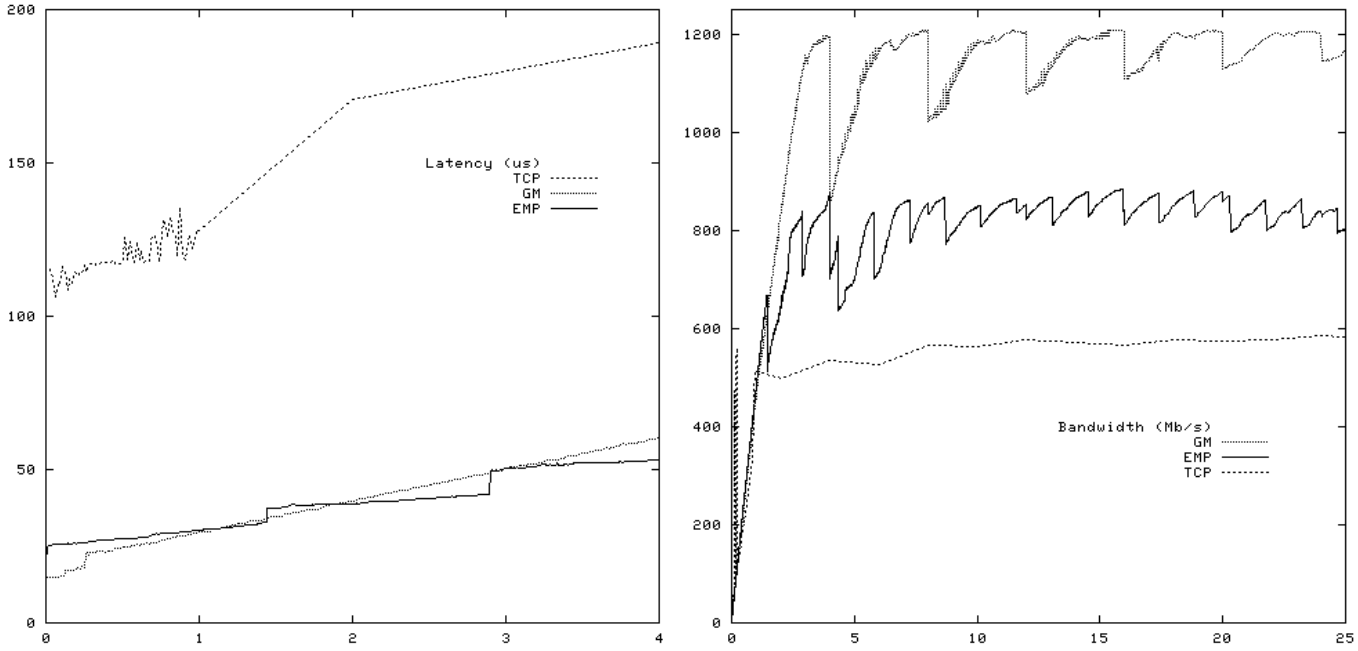

Figure 5: Latency and bandwidth comparisons. The quantity measured on the abscissa is message size in kilobytes, for all three plots.

which has a 64-bit 66 MHz PCI bus, and unmodified Linux version 2.4.2. Our NICs are Netgear 620 [12], which have 512 kB of memory. The machines are connected back-to-back with a strand of fiber. For the Myrinet tests, we used the same machines with LANai 7.2 cards connected with a single LAN cable. We do not measure switch overheads, but expect them to be roughly 2–4 us for ethernet and less than 1 us for Myrinet. All tests using ethernet were performed with a maximum transfer unit (MTU) of 1500 bytes. The TCP tests used 64 kB socket buffers on both sides; more buffering did not increase performance.

5.2 Results and Discussion

Figure 5 shows plots for the latency and bandwidth for each of the three systems as a function of the message size. The latency is determined by halving the time to complete a ping-pong test, and the bandwidth is calculated from one-way sends with a trailing return acknowledgement. Each test is repeated 10 000 times to average the results.

The latency for TCP over Gigabit Ethernet is much higher than the other two systems due to the overhead of invoking the operating system for every message. GM and EMP both implement reliability, framing, and other aspects of messaging in the NIC. In the very small message range under 256 bytes, GM imposes a latency which is up to 10 us lower than that of EMP, but the per-byte scaling of both EMP and GM is similar and message latencies are comparable.

The theoretical wire speed of our test version of Myrinet is 1200 Mb/s, and GM delivers very near to that limit. Gigabit ethernet, though, is limited at 1000 Mb/s, but EMP only delivers 80% of that limit. The packet size used in GM is 4 kB, and it exhibits the same periodic performance drop as does EMP, since that coincides with the host page boundary. The jumps at the frame size of 1500 bytes where the NIC must process an extra frame are clearly visible, and jumps at the page size of 4096 bytes where the NIC must start an extra DMA are also visible. The bandwidth of TCP never exceeds about 600 Mb/s, and is consistently spiky at the low message sizes, probably due to the kernel aggregating many small messages into a single IP packet.

One further point of comparison is the cost to the host of sending or receiving a message. As was emphasized earlier, previous work by others suggests that the CPU overhead of sending a message impacts the overall performance of a parallel application more than does the network latency. We measure CPU utilization by running a synthetic MPI application which mimics a common cycle seen in many user codes [9]. Each iteration of a measurement loop includes four steps: post receives for expected incoming messages, initiate asynchronous sends, perform computational work, and finally wait for messages to complete. As the amount of computational work is increased, the host CPU fraction available to message passing decreases, resulting in the declining graphs in Figure 6. The exact timings for each operation of an iteration are shown in Table 2.

The advantages of offloading the work of messaging onto the NIC are clearly seen where the available EMP bandwidth remains flat with increasing host CPU load, while both GM and TCP slope downwards when any CPU cycles are spent on activities other than moving bytes through the network. Although the times in GM and EMP to post a send or receive are quite similar, it is the wait time which is much more expensive in GM (and in TCP). In all three systems, the call to MPI_Waitall waits for incoming messages to arrive, but in GM and TCP, the call to further performs a

Table 2: Asymptotic message passing function timings (us), per function call for 10kB messages in batches of five messages at a time.

Function	MPI Call	TCP	GM	EMP
Post transmit	MPI_Isend	137	66	53
Post receive	MPI_Irecv	4	1	9
Wait	MPI_Waitall	126	123	3
Total		266	190	65

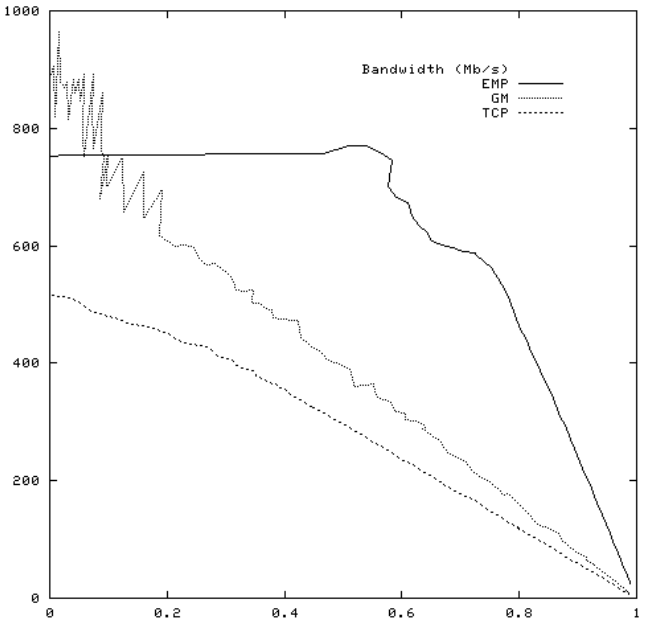

Figure 6: Throughput as a function of CPU fraction used by user code for 10kB messages.

memory copy from the network buffers into the desired final location in the memory of the application. In EMP, however, the NIC itself DMAs directly into the application while the host CPU is performing other work.

As the computational demand on the host increases toward the right side of Figure 6, the bandwidth achieved by EMP as a function of computational work falls toward zero. This is due to the network being starved for work. The NIC has completed sending and receiving its messages for that iteration before the host completes its corresponding computational work. Since we measure throughput by dividing the sum of user payload sent or received by the *total* amount of time consumed in the loop, all graphs must end up at zero.

6. CONCLUSIONS AND FUTURE WORK

Our objective was to produce a reliable, low latency and high throughput, NIC-driven messaging system for Gigabit Ethernet. We came across numerous design challenges for accomplishing this which were overcome by redistributing the various functions of the messaging system in a novel way. We moved all the protocol related functions to the NIC and utilized the processing capabilities of the Alteon NIC, removing the kernel altogether from the critical path. All the work of moving the multiple frames of the message, ensuring reliability, demultiplexing incoming frames, and addressing are performed by the NIC.

The NIC handles all queuing, framing, and reliability details asynchronously, freeing the host to perform useful work. We obtained a latency of 23 us for 4 byte message, 36 us for 1500 bytes; and a throughput of 880 Mb/s for 4 kB messages. From these results, we can conclude that EMP holds tremendous potential for high performance applications on Gigabit Ethernet.

We plan to explore the design space of EMP in terms of NIC *vs.* host. Currently we have aimed to implement the entire protocol processing at the NIC. In future we want to know how to optimize the processing in the best possible way by distributing the work across the host and the NIC. This gains importance in the light of current host processing power available to us.

We have an initial MPI implementation using the ADI2 interface of MPICH which performs all the basic functionality of MPI 1.2, but some features are lacking (Probe and Cancel). We plan to extend and improve our MPI support, as well as provide support for security and for multiple concurrent applications on a single node.

7. ACKNOWLEDGEMENTS

We would like to thank the Sandia National Labs for supporting this project (contract number 12652 dated 31 Aug 2000). We would also like to thank the graduate students and the faculty of the NOWLab at the Ohio State University for giving us their invaluable advice on numerous issues related to the project. The anonymous referees for this paper also deserve our thanks for their useful feedback and recommendations for improving the quality of this paper.

8. REFERENCES

[1] R. Bhoedjang, K. Verstoep, T. Ruhl, H. Bal, and R. Hofman. Evaluating design alternatives for reliable communication on high-speed networks. In *Proceedings of ASPLOS-9*, November 2000.

[2] N. Boden, D. Cohen, and R. Felderman. Myrinet: a gigabit per second local-area network. *IEEE Micro*, 15(1):29, February 1995.

[3] M. Boosten, R. W. Dobinson, and P. D. V. van der Stok. MESH: Messaging and scheduling for fine-grain parallel processing on commodity platforms. In *Proceedings of PDPTA*, June 1999.

[4] G. Chiola and G. Ciaccio. GAMMA, http://www.disi.unige.it/project/gamma.

[5] B. Chun, A. Mainwaring, and D. Culler. Virtual network transport protocols for Myrinet. Technical Report CSD-98-988, UC Berkeley, 1998.

[6] C. Csanady and P. Wyckoff. Bobnet: High-performance message passing for commodity networking components. In *Proceedings of PDCN*, December 1998.

[7] P. Gilfeather and T. Underwood. Fragmentation and high performance IP. In *CAC Workshop*, October 2000.

[8] Infiniband. http://www.infinibandta.org.

[9] B. Lawry, R. Wilson, and A. B. Maccabe. OS bypass implementation benchmark. http://www.cs.unm.edu/~maccabe/SSL, 2001.

[10] R. Martin, A. Vahdat, D. Culler, and T. Anderson. Effects of communication latency, overhead, and bandwidth in a cluster architecture. In *Proceedings of ISCA*, June 1997.

[11] MVIA. http://www.nersc.gov/research/FTG/via, 1998.

[12] Netgear. http://www.netgear.com.

[13] S. Pakin, M. Lauria, and A. Chien. High performance messaging on workstations: Illinois Fast Messages (FM) for Myrinet, 1995.

[14] P. Pietikainen. Hardware acceleration of Scheduled Transfer Protocol. http://oss.sgi.com/projects/stp.

[15] I. Pratt and K. Fraser. Arsenic: a user-accessible gigabit ethernet interface. In *Proceedings of Infocom*, April 2001.

[16] S. Sumimoto, H. Tezuka, A. Hori, H. Harada, T. Takahashi, and Y. Ishikawa. High performance communication using a gigabit ethernet. Technical Report TR-98003, Real World Computing Partnership, 1998.

[17] VI. http://www.viarch.org, 1998.

[18] 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 the 15th ACM Symposium on Operating Systems Principles*, December 1995.