# Low-Latency Message Passing Over Gigabit Ethernet Clusters

Keith Fenech

Department of Computer Science and AI,
University of Malta

**Abstract.** As Ethernet hardware bandwidth increased to Gigabit speeds it became evident that it was difficult for conventional messaging protocols to deliver this performance to the application layer. Kernel based protocols such as TCP/IP impose a significant load on the host processor in order to service incoming packets and pass them to the application layer. Under heavy loads this problem can also lead to the host processor being completely used up for processing incoming messages, thus starving host applications of CPU resources. Another problem suffered by inter-process communication using small messages is the latency imposed by memory-to-memory copying in layered protocols as well as the slow context switching times in kernel-level schedulers required for servicing incoming interrupts. All this has put pressure on messaging software which led to the development of several lower latency user-level protocols specifically adapted to high-performance networks (see U-Net[18], EMP[16], VIA[3], QsNET[15], Active Messages[19], GM[13], FM[14]).
The aim of this paper is to investigate the issues involved in building high performance cluster messaging systems. We will also review some of the more prominent work in the area as well as propose a low-overhead low-latency messaging system to be used by a cluster of commodity platforms running over Gigabit Ethernet. We propose to use the programmable Netgear GA620-T NICs and modify their firmware to design a lightweight reliable OS-bypass protocol for message passing. We propose the use of zero-copy and polling techniques in order to keep host CPU utilization to a minimum whilst obtaining the maximum bandwidth possible.

## 1 Introduction

The performance of off-the-shelf commodity platforms has increased multi-fold over the past two decades. The same can be said for the continuous increase in demand for computational speed by modern applications, especially in areas involving scientific and engineering problem simulations and numerical modeling. In most cases such computation can be performed by multi-threaded applications running in parallel on separate processors. The increased availability of high bandwidth networking hardware which is currently capable of transmitting at gigabit/sec speeds has enabled the development of Networks of Workstations (NOWs) [5] which are turning out to be more affordable replacements to single large-scale computers.

The inclination towards the use of clustered computing for high performance computation has put forward new requirements that cannot be delivered by conventional communication systems used in commodity platforms. The major challenge in using NOWs for parallel computation is that the communication architecture should be as transparent as possible to the user-level applications and must not act as a performance bottleneck. Thus, the aim is to provide a high-throughput and low-latency message passing mechanism that enables fine grain multi-threaded applications to communicate efficiently whilst keeping processing overhead on host machines to a minimum.

Although commodity networks are still dominated by IP over Ethernet, these protocols were not intended for networks running at gigabit speeds due to several inefficiencies that will be discussed later. This led to the development of other interconnect architectures such as Myrinet [2], SCI [10], Giganet cLAN [7], Quadrics [15] and Infiniband [1] which were targeted towards achieving higher bandwidth reliable communication without incurring the processing overheads of slower protocols. Our work will be geared towards commodity platforms running over a Gigabit Ethernet architecture, since these are one of the most common and affordable cluster architectures available.

## 2 High Performance Networking Issues

Various performance issues arise when utilizing the standard Ethernet framework for gigabit communication within a cluster of commodity platforms. Although modern Ethernet hardware promises gigabit line-rates, this performance is hardly ever obtainable at the application layers. This is due to several overheads imposed by interrupt generation [8] as well as conventional layered communication protocols such as TCP/IP which although provide reliable messaging, also add significant protocol processing overheads to high bandwidth applications. First of all we will take a look at the steps involved in sending messages over a network using conventional NICs and generic protocols such as TCP/IP.

### 2.1 Conventional Ethernet NICs

Conventional network cards are cheap and usually characterised by little onboard memory and limited onboard intelligence which means that most of the protocol processing must be accomplished by the host. When the Ethernet controller onboard the NIC receives Ethernet frames, it verifies that the frames are valid (using CRC) and filters them (using the MAC destination address). When a valid network packet is accepted, the card generates a hardware interrupt to notify the host that a packet has been received. Since hardware interrupts are given a high priority, the host CPU suspends its current task and invokes the interrupt handling routine that will check the network layer protocol header (in our case being IP). After verifying that the packet is valid (using a checksum) it also checks whether that particular IP datagram is destined for this node or whether it should be routed using IP routing tables. Valid IP datagrams are stored in a buffer (IP queue) on host memory and any fragmented IP datagrams are reassembled in the buffer before calling the appropriate transport layer protocol functions (such as TCP/UDP). Passing IP datagrams to upper layer protocols is done by generating a software interrupt and usually this is followed by copying data from the IP buffer to the TCP/UDP buffer.

### 2.2 Problems with Conventional Protocols

Since traditional operating systems implement TCP/IP at the kernel level, whenever an interrupt is generated the host would incur a vertical switch from the user to the kernel level in order to service the interrupt. This means that for each packet received the host would experience several switching overheads. Also due to the layered protocol approach, each packet would require several memory-to-memory buffer copies before it reaches the application layer. Both memory-to-memory copies as well as context switching (between kernel and user space) are expensive to the host CPU which means that the higher the rate of incoming packets, the more host processing power is required to service these packets.

Servicing these interrupts on a 100Mbps fast ethernet network would be manageable by the receiving host using around 15% CPU utilization, however when projecting this to a 1Gbps network using the same host we would easily get to 100% CPU utilization before reaching 600Mbps [8]. This weakness in traditional protocols can lead to a *receiver livelock* scenario which can be easily exploited to cause denial of service attacks. This scenario is brought about since hardware interrupts generated by incoming packets are given a higher priority than software interrupts generated by higher layer protocols. Also user level applications which are waiting for the data are given the lowest priority. This means that under high network load the applications processing the incoming data would be continuously interrupted by incoming packets and thus would not have enough processing time available to service the incoming data. Unserviced packets would remain waiting in the respective protocol queues and this could easily result in the overflow of protocol buffers. In case of overflow, all newly received packets would be discarded, however these would still be using up host processing time due to the interrupts they generate. This would result in the host machine consuming all its processing resources for servicing and discarding incoming packets while starving user applications from CPU time.

### 2.3 Current Solutions

There have been several attempts to achieve better throughput while maintaining low CPU utilization. U-Net [18] was one of the first systems that removed the kernel from the communication path and enabled user-level access to the communication hardware which in their case worked over ATM. This OS-bypass mechanism was also exploited by Myrinet GM [13] which uses a technique to allocate and register a common chunk of host memory which is then used by the hardware for sending/receiving messages to/from user-space. Myrinet [2] interfaces have an onboard processor that continuously loops through what is referred to as a *Myrinet Control Program* that allows user-applications to communicate with the hardware without using system calls, thus eliminating the need for vertical switches during communication.

Projects like U-Net also make it possible for protocol processing to move into user-space. Thus, unlike traditional kernel based protocols, U-Net delivers flexibility to applications that want to customize their own protocols and interfaces tailored to application specific requirements. U-Net also moves all buffer management to user-space which gives the illusion that each process owns the network, however it also gives rise to issues dealing with data protection between processes using the same network. U-Net also supports a zero-copy architecture in which data can be passed between the application data structures and the network interface without any intermediate buffering. In order not to involve the kernel in the critical path, U-Net uses an interrupt-free mechanism and transmits data by pushing message descriptors onto a queue. A polling mechanism enables the NIC to retrieve the data from the buffer and transmit it over the network. Receiving messages is done in a similar way using events.

Other performance enhancements can be achieved through interrupt coalescing so that hardware interrupts are not generated for each received packet but packets are batched together and passed to the host in larger chunks, thus incurring less DMA startup overheads. Interrupts can be generated when an incoming buffer treshold is exceeded or when a timer expires or else using a hybrid approach between these two. On switches and NICs that support a non-standard maximum transmission unit (MTU) we can instruct the NIC to use larger MTUs (9KB Jumbo frames) instead of the standard 1.5KB Ethernet frames in order to reduce the number of transmitted Ethernet frames as well as reduce the interrupts generated when using applications requiring high throughput.

The techniques used by U-Net, Myrinet GM and other architectures such as QsNet [15] were only possible by utilizing programmable network interface cards. These type of cards having one or more onboard processors and memory which make it possible for them to interact directly with

user-applications and thus offload message passing overheads from the host. Another overhead that may be alleviated from the host processor is that of calculating checksums for each transmitted network packet. Making use of the NIC CPU it is possible to instruct the firmware on the card to perform these checksums before passing them on to the host application.

One of the first host-offloading systems to be developed for the Gigabit Ethernet architecture was EMP [16]. This was the first NIC-level implementation of a zero-copy OS-bypass messaging layer that used programmable NICs to perform the entire protocol processing onboard the NIC. EMP made use of the NIC's DMA facilities for transferring packets to/from the host, thus allowing the CPU to remain available for application processing.

Several issues arise when using DMA to copy data directly from the user-space to the NIC memory or vice-versa without any kernel interaction. First of all user applications make use of virtual memory addresses provided by the OS whilst the DMA engine on the NIC requires physical memory addresses in order to perform data transfers to/from the host. Thus before using DMA, the architecture is required to implement some sort of mechanism for mapping virtual to physical addresses. One way of doing this is by keeping a copy of the kernel page-tables on the NIC however due to the limited memory onboard the NIC this would impose constraints on the amount of memory that can be mapped. Another option is to have the user application inform the NIC about the physical addresses of the pages involved in the transfer. Since user-space does not have access to physical memory addresses, these would have to be retrieved from the kernel through a system call. This is the technique used by Shivam *et al.* in EMP who also argue that it is not too computationally expensive to perform such system calls for translating memory addresses. Another issue arising when DMAing directly to/from user-space is that modern operating systems use complex Virtual Memory management mechanisms in order to allow applications to use more memory than physically available. This allows memory pages to be paged out from physical memory and stored in virtual memory automatically by the OS. This would provide a problem for the NIC DMA engine if pages are removed from memory whilst a DMA transfer is taking place. To solve this EMP make use of UNIX system calls that instruct the OS not to allow paging for specific memory regions. Such memory locking would allow zero-copy message transfers between the NIC and user-space. Tezuka *et al.* [17] also implemented a zero-copy message transfer in their Myrinet PM network driver which they call *Pin-down Cache*. Their implementation provides an API that allows the user application to pin-down and release physical memory to be used for zero-copy messaging. However pinned down areas are managed by the system and are reused in order to reduce the number of pin-down and release primitives as well as to protect physical memory from being exhausted by malicious users.

Another issue to consider when using DMA for data transfers is that the pages involved in the transfer are not always aligned in one contiguous block which compels a problem for the DMA engine which uses physical address. To avoid expensive buffer alignments prior to using DMA data transfers we can make use of scatter/gather techniques where fragmented packets can be retrieved from different parts in memory without having to perform additional memory copies to align the buffer data into a contiguous chunk. This can be accomplished by performing multiple DMA transfers for each packet.

Each time a transfer occurs through the PCI bus, a startup latency is inherently induced by the bus, however once the transfer is started there are no further overheads unless the bus operations are interrupted by the host CPU or other connected devices [9]. Consequently increasing the size of data transfers over the PCI bus would lead to a higher bandwidth due to less setup latency as noted in tests done by Cordina [4] and Wadge [20]. On similar lines, Gilfeather and Underwood [8] proposed transferring larger MTUs (64K) over the PCI bus in order to reduce interrupt overhead. This was done by modifying the NIC firmware to enable fragmentation of large datagrams into smaller frames on the NIC before being transmitted over the network. On the receiving side, the NIC re-assembles the datagrams before sending them through the host PCI bus.

### 2.4 Short messaging

Traditional network performace is usually measured by the bandwidth achieved for continuous streams, however many applications relying on short messages are more sensitive to the communication latency which can be described as the round-trip time for a message to travel between two processes running on networked nodes. This is the case with fine-grain multi-threaded applications which communicate very frequently by sending small messages usually under 1KB. There have been several attempts to minimize the latency incurred by small message passing over a cluster. One such attempt is Illinois Fast Messaging FM [14] for Myrinet which offers a low-latency reliable messaging layer designed specifically for sending small messages over tightly coupled clusters.

Intel[1], Microsoft[2] and Compaq[3] have joined forces in an effort to standardize an architecture for the interface between computer systems and high performance hardware. First drafted in 1997, the Virtual Interface Architecture (VIA) [3] aims to improve the performance of distributed applications by reducing the software processing required to exchange messages. The VI architecture can be said to have been built upon U-Net, FM and Active Messages [19] and provides each consumer process with a protected and directly accessible virtual interface (VI) to the shared network hardware. The VIA only uses the OS to setup data structures and mappings for the user-level network (ULN) so the kernel can be removed from the critical path. This implies that the network adapter must have a certain degree of intelligence in order to perform all the data transfer scheduling and de/multiplexing usually performed by the OS. VIA uses a system of buffer descriptors and doorbells to pass messages between the NIC and user-space. This event driven system uses polling to avoid interrupt and system call overheads for message passing.

### 2.5 The Alteon Acenic Tigon 2 NIC

The work done by Gilfeather and Underwood [8], EMP [16], Cordina [4] and the tests done by Wadge [20] were all implemented by using Alteon Networks' Acenic[4] [12, 11] NIC which is a PCI programmable Gigabit Ethernet card having two onboard 32-bit/88 MHz RISC processors. Unlike ROM based NICs, the firmware is uploaded onto the card during initialization which means that it can be modified and reloaded onto the card at any time. The card also has 1MB of SRAM which is used for holding the firmware as well as any other data buffers. The Alteon Acenic is also equipped with 2 DMA engines and supports a list of extra features such as the ability to perform onboard IP/TCP/UDP checksum calculations which is useful for alleviating such a load from the host processor.

One of the reasons why the Alteon Acenic is popular is due to the fact that Alteon Websystems have provided complete documentation for their NIC API and the Tigon 2 firmware is available as open source. This allows us to optimize the firmware for non-standard protocols in search for optimal performance results. Unfortunately in the early 2000 the Alteon Acenic was discontinued and Alteon was taken over by 3Com who replaced the Tigon 2 by an improved design; the Tigon 3. Unlike Alteon, 3Com decided not to provide documentation for the new NIC which makes it very difficult to reprogram the card.

[1] http://www.intel.com/

[2] http://www.microsoft.com/

[3] http://www.compaq.com/

[4] Also known as the Tigon 2

## 3 Our Proposal

Since the Alteon Acenic is no longer in production it was very difficult to acquire a pair of these cards for experimenting with customizable firmware, however we did find the Netgear GA620-T which is a similar model that makes use of the same Tigon 2 chipset. One of the major differences between the two is that the GA620-T has only 512KB of onboard memory compared to the 1MB of the Alteon Acenic. Using the documentation provided by Alteon for their Tigon 2 we will attempt to use the Netgear programmable NICs to develop an architecture for high-throughput low-latency message passing between processes running over a Gigabit Ethernet cluster.

Our primary aim is to eliminate any OS interaction during communication by building a user-level network similar to the VI architecture that gives user-level applications direct access to the NIC. This would involve creating a pinned down area in host memory that would be shared between the NIC and the host processes. Our system will use polling and events in replacement of interrupts in order to keep host CPU utilization to a minimum. We will use the Netgear's DMA facilities together with a zero-copy technique in order to reduce latency for small messages.

We will also modify the firmware to experiment with different size MTU's both for DMA and Ethernet frames as well as enable large PCI transfers in order to increase the overall throughput for applications requiring high bandwidth. We plan to design a lightweight but reliable user-level protocol for fast messaging that takes into consideration the reliability of the underlying physical network in order to provide the highest bandwidth possible with the lowest CPU utilization. This would be aided by offloading part of the checksum and protocol processing onto the NIC as far as the card's onboard CPUs allow us.

The final goal would be that of providing an API that can be seamlessly integrated with a user-level thread scheduler such as SMASH [6] developed by Debattista. This would allow users to develop fine-grain multi-threaded applications whose threads can be efficiently scheduled over SMP nodes connected on a gigabit cluster. Our designs will be implemented over the Linux platform.

## 4 Conclusion

We have highlighted various bottlenecks that arise when using conventional networking protocols for high-performance computing as well as illustrated several issues involved in developing high-performance communication protocols. We have analysed several attempts by previous projects aimed towards increasing throughput and decreasing latency for message passing over different gigabit network architectures. Finally we have stated our aim to use Netgear GA620-T programmable NICs together with the Linux OS in order to provide an API that allows efficient messaging between multi-threaded user-level applications running in a distributed environment over a cluster of SMPs.

## References

1. Infiniband Trade Association. Infiniband Architecture. http://www.infinibandta.org/ibta/.
2. Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, 1995.
3. Microsoft Compaq, Intel. *Virtual Interface Architecture Specification*, draft revision 1.0 edition, December 1997.

4. Joseph Cordina. High Performance TCP/IP for Multi-Threaded Servers. Master's thesis, University of Malta, 2002.
5. D. Culler, A. Arpaci-Dusseau, R. Arpaci-Dusseau, B. Chun, S. Lumetta, A. Mainwaring, R. Martin, C. Yoshikawa, and F. Wong. Parallel Computing on the Berkeley NOW. In *Ninth Joint Symposium on Parallel Processing*, 1997.
6. Kurt Debattista. High Performance Thread Scheduling on Shared Momory Multiprocessors. Master's thesis, University of Malta, 2001.
7. Emulex. clan. http://www.emulex.com/ts/legacy/clan/index.htm.
8. Patricia Gilfeather and Todd Underwood. Fragmentation and High Performance IP. University of New Mexico.
9. PCI Special Interest Group. *PCI Local Bus Specification*, revision 2.1 edition, June 1995.
10. Davib B. Gustavson. The Scalable Coherent Interface and Related Standards Projects. *IEEE Micro*, 12(1):10–22, 1992.
11. Alteon Networks Inc. *Tigon/PCI Ethernet Controller*, revision 1.04 edition, August 1997.
12. Alteon Networks Inc. *Gigabit Ethernet PCI Network Interface Card, Host/Nic Software Interface Definition*, revision 12.4.13 edition, July 1999.
13. Myricom Inc. Myrinet GM – the low-level message-passing system for Myrinet networks.
14. Scott Pakin, Mario Lauria, and Andrew Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet. 1995.
15. Fabrizio Petrini, Wu chun Feng, Adolfy Hoisie, Salvador Coll, and Eitan Frachtenberg. Quadrics Network (QsNet): High-Performance Clustering Technology. In *Hot Interconnects 9*, Stanford University, Palo Alto, CA, August 2001.
16. Piyush Shivam, Pete Wyckoff, and Dhabaleswar Panda. EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing. 2001.
17. H. Tezuka, F. O'Carroll, A. Hori, and Y. Ishikawa. Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication. pages 308–315.
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 fifteenth ACM symposium on Operating systems principles*, pages 40–53. ACM Press, 1995.
19. Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser. Active Messages: A Mechanism for Integrated Communication and Computation. In *19th International Symposium on Computer Architecture*, pages 256–266, Gold Coast, Australia, 1992.
20. Wallace Wadge. Achieving Gigabit Performance on Programmable Ethernet Network Interface Cards. University of Malta, 2001.