

A Quantitative Study on the Communication Performance of Myrinet Network Interfaces

Mark Baker and Hong Ong
University of Portsmouth
Portsmouth, UK

March 15, 2002

Abstract

This paper presents a series of experiments for evaluating the communication performance of Myrinet network adaptors. In particular, this paper studies the impact that arises from the interactions between the Myrinet network adaptors and host computer systems on communications performance. We report and analyze the communication performances of GM, the low-level messaging API for Myrinet, and other network protocols such as IP and MPI that are built on top of GM. All the performance results are obtained using a Pentium-based cluster running the Linux operating systems and interconnected by the Myricom's *LANai9 (M3M-PCI64B)* host adapters.

ABSTRACT	1
1 INTRODUCTION.....	3
2 AN OVERVIEW OF MYRINET 2000 TECHNOLOGY	4
3 TESTING ENVIRONMENT.....	6
4 RESULTS OF BENCHMARKING.....	7
4.1 Memory Performance.....	7
4.2 PCI Performance	8
4.3 GM LogP Performance	8
4.4 TCP/IP over GM Performance.....	12
4.5 MPICH over GM Performance	15
5 CONCLUDING REMARKS	16
REFERENCES.....	18

1 Introduction

Beowulf clusters are a cost-effective means of delivering significant amounts of computational power [1]. In a cluster environment, a number of Commercial-Off-The-Shelf (COTS) computers are combined to act as a single, large multiprocessor system. With high-speed networks¹ and communication libraries, Beowulf clusters have been used to demonstrate significant gains in parallel processing.

However, it has been shown [2][3] that applications have not been able to take full advantage of these high-speed networks due to, among other factors, the processing overhead of legacy protocols such as TCP/IP. Specifically, these protocols require that all network accesses be through the operating system, which contributes a significant overhead to both the transmission path (i.e., a system call and data copy) and the reception path (i.e., an interrupt, a system call, and a data copy). To overcome the inherent overheads caused by the legacy protocol processing, significant research efforts have been carried out. Among these efforts, two well-known techniques are used to improve the communication performance are the user-level network protocol and the zero-copy protocol schemes.

The user level network protocol scheme, also known as OS-by-pass, exposes a lower-level abstraction of the network device than is normally provided. Applications are given their own virtual interface so that they can manipulate the network queue buffers for transmission and reception. Bypassing the conventional operating system mechanisms allows applications to reduce the latency and per-packet overhead of using the network. This is because the overhead associated with the system calls is avoided and the extra data copies are minimized. In addition, better communication scheduling can be achieved since the network traffic can be predicted more accurately with either mathematical or probabilistic models. Examples of the early user-level network protocols used in these user-level networks are U-Net [4], Fast Message (FM) [5], Active Message (AM) [6], GM [7]. The design and development of these user-level network efforts have also lead to an industry standard known as Virtual Architecture Interface (VIA) [8][9] More recently, VIA has been included in the InfiniBand Architecture (IBA) [10][11].

Another technique used to improve communication performance is the zero-copy protocol scheme. Zero-copy means that applications are given direct buffer management at the network layer. Unfortunately, this optimization is awkward to implement with the current BSD sockets API specification. The semantics of the BSD socket interface states that the user data buffer could be reused immediately after the system call (i.e., `write`) has fully copied the data to the system buffer. However, with zero-copy protocol, the only data buffer is the user data buffer. This implies that the user data buffer may not be ready for re-use immediately. To preserve the existing semantics, a common approach is to have memory pages containing the data marked as Copy-On-Write (COW) by the Memory Management Unit (MMU). This essentially prevents the application from

¹ Examples of high-speed networks are ATM, Fibre Channel, ISDN, Gigabit Ethernet, and etc.

writing to the pages. If the application attempts to overwrite the data, the OS creates a copy of the original data and transparently maps the copy into the user virtual address space. After the data has been acknowledged the COW mapping can be released. Avoiding the copy at the receiver side is much more complicated since the network adaptor requires knowledge about the protocol states in order to determine where the data should be placed. In addition, memory pages to which data is to be placed must be locked to prevent it from being paged out to disk. Smith et al. [12] outline a variety of approaches for implementing the zero-copy protocol. Experimental implementations could be found in Trapeze [13] based on Myrinet [14], Linux 2.4 kernel, Solaris, and FreeBSD. Currently, there is no standard for implementing the zero-copy protocol.

It is not the intention of this paper to provide a comparative analysis of these various user-level network and zero-copy implementations or a comparative analysis between these two techniques. Instead, this paper aims to provide a quantitative study of the interaction between the Myrinet network adapters and the host systems. The Myrinet network adaptor was chosen because it supports both the user-level network and the zero-copy protocols. By investigating the interaction between the network adaptor and the host system, this paper reveals aspects of behaviour that are critical for the architectural design of an efficient communication system. The rest of this paper is organized as follow: Section 2 provides an overview of Myrinet 2000 technology. The testing environment is presented in Section 3. Section 4 presents the performance results and identifies the issues that are most directly impact system performance. The paper is concluded in Section 5.

2 An Overview of Myrinet 2000 Technology

Myrinet-2000, developed by Myricom, is a proprietary network technology and is compliant to the Physical and Data Link layer defined in the ANSI/VITA 26-1998 standard. Myrinet is a switched, Gigabit per second network that is widely used in Beowulf clusters and embedded system. A Myrinet-2000 network [15] is composed of crossbar switches and network adaptors. Network adaptors are connected to the switches through point-to-point duplex links and the crossbar switches can be interconnected in an arbitrary topology.

The basic building block for the Myrinet-2000 switche is a 16-port crossbar (XBar16). Currently the maximum number of hosts supported within a single unit (enclosure) is 128, using 24 XBar16 switches. The topology used to interconnect these switches is a full-bisection Clos network. By applying the same Clos network principle, Myrinet-2000 can scale up to 8192 hosts that can potentially offers a network bisection bandwidth in the order of Terabits per second. Data packets are wormhole routed from one network adaptor to another through a series of crossbar switches enabling a latency of approximately half a microsecond. Flow control is achieved by inserting STOP and GO control bytes into the opposite channel of a link by the receiver side to stop or restart data transmission on the sender side. As a result, Myrinet would not normally drop packets unless the receiver fails to drain the network. Error control is accomplished by computing an 8-bit cyclic-redundancy check (CRC-8) on the entire packet including packet header.

The CRC-8 is then carried in the packet trailer and recomputed in each network stages. Thus, data packet entering a host interface with a non-zero CRC-8 indicates transmission errors.

The Myrinet-2000 network adaptor is a programmable communication device that provides an interface to the System Area Network (SAN). It consists of three major components (see Figure 1):

1. A custom VLSI (LANai) chip,
2. A synchronous static RAM memory,
3. A PCI Bridge and a DMA controller.

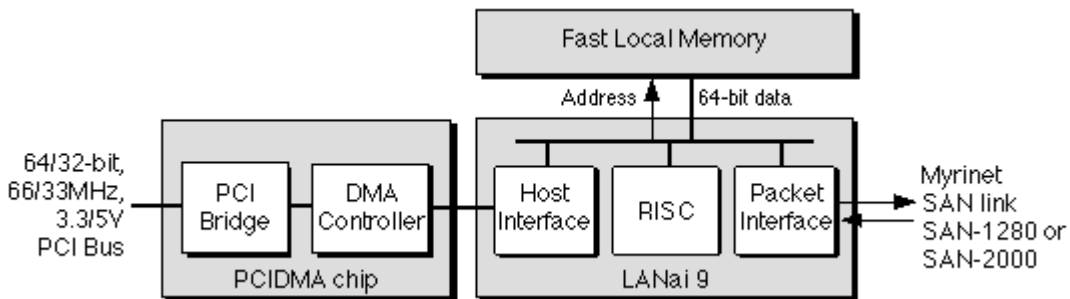


Figure 1: LANai9 Architecture

Like other modern network adaptors, Myrinet adaptors have a programmable network interface processor known as LANai9. LANai9 is a 32-bit RISC processor that operates at up to 133MHz for the PCI64B interfaces, or at up to 200MHz for the PCI64C interfaces. Using the Myrinet Control Program (MCP), which is stored in the onboard static RAM (SRAM), LANai9 controls the data transfer between the host and the network (through the host and packet interface), performs data buffer management (through memory interface), and maintains network mapping and monitoring. The benefit of a programmable network processor is that it enables researchers to explore many protocol design options.

The onboard Myrinet-2000 SRAM size ranges from 2Mbytes to 8 Mbytes and operates at the same clock speed as LANai9, i.e., at up to 133 MHz for the PCI64B interfaces or at up to 200MHz for the PCI64C interfaces. This suggests that the maximum attainable bandwidth is approximately 1,064 Mbytes/s and 1,600 Mbytes/s for the PCI64B and PCI64C respectively. The SRAM is accessible from both the onboard local bus (LBUS) and the external system bus (EBUS). LBUS and EBUS are both 64-bit wide but the LBUS is clocked at twice the chip-clock speed, permitting two LBUS memory accesses per clock cycle.

To increase the data transfer rate, a Myrinet-2000 network adaptor is equipped with three DMA engines. Two DMA engines are associated with the packet interface: one for receiving packets and one for sending packets. The third DMA engine is used for data transfer between the SRAM and the host system memory through the host interface. Like most systems that support DMA, the on board memory can be mapped into user space

and is thus accessible directly to user processes. This mapping is performed in two steps: the OS first maps the NIC address space into kernel space and then, on a request by user, the kernel region is mapped into user space. This memory mapping technique is commonly known as “memory pinning”. In order to support zero-copy APIs efficiently, the DMA operations can be performed with arbitrary byte counts and byte alignments. Additionally, the DMA engine also computes the IP checksum for each transfer and provides a "doorbell" signaling mechanism that allows the host to write anywhere within the doorbell region, and have the address and data stored in a FIFO queue in the local memory.

The host processor can also access the Myrinet-2000 SRAM through the programmable input/output (PIO) interfaces. With PIO, the host processor reads the data from the host memory and writes it into the Myrinet-2000 SRAM. This mode of data transfer typically results in many PCI I/O bus transactions. Although Myrinet-2000 PCI64 interfaces are capable of sustained PCI data rates approaching the limits of the PCI bus, the network performance greatly depends on the data transfer rate of the host’s memory and PCI-bus implementation.

All Myricom software for the PCI64 family of interfaces is based on the GM Myrinet Control Program (MCP) and the GM API. GM is a lightweight message-based communication system for Myrinet designed by Myricom. Other efficient middleware implementations over Myrinet are available from Myricom and from third parties. A detail description of the Myrinet 2000 technology can also be found in [15][16].

3 Testing Environment

The testing environment for collecting the performance results consists of a cluster of two Pentium 200MHz PCs using the Intel 430VX chipset. The PCs were equipped with 64Mbytes of 33MHz RAM and 512Kbytes cache. The cluster operating system was the Red hat 7.1 Linux with kernel version 2.4.9-21. Myrinet LANai9 host adaptors (PCI64B) are installed in the 32bit/33MHz PCI slot and were connected back-to-back using Myrinet M3S-CB-3M cables. The host adaptor is equipped with 2Mbytes of memory.

The firmware used for testing LANai9 was the GM 1.5. Apart from GM, MPICH-GM version 1.2.1.7b and VI-GM version 0.9 for Linux x86 were also installed. MPICH is a portable implementation of MPI developed by Argonne National Laboratory. MPICH-GM is a port of MPICH on top of GM. VI-GM is a port of VIA on top of Myrinet.

To measure the performance of the LANai9 several test programs from the GM 1.5 distribution and the NetPIPE network benchmark were used to collect the attainable bandwidth and latency at each layer of the communication protocol stacks.

4 Results of benchmarking

4.1 Memory Performance

It is well known that the computer memory performance affects the performance of high-speed network. In order to quantify how the memory performance affects the measurements of Myrinet, a series of memory tests were performed using the HINT [17][18] benchmark. HINT (Hierarchical INTEgration) is a performance-benchmarking tool developed at the Scalable Computing Laboratory (SCL) of the Ames Laboratory of the U.S. Department of Energy (DOE). Unlike traditional benchmarks, HINT neither fixes the size of the problem nor the calculation time, instead uses a measure called QUIPS (Quality Improvement Per Second) . This enables HINT to display the speed for a given machine specification and problem size. This speed typically decreases as the calculation moves from cache access to main memory access and finally to virtual memory disk accesses. These transitions are easily visible in HINT generated data results. Thus, various points on the HINT graph correspond to various memory regimes and may be viewed as corresponding to some degree to other more conventional benchmarks such as Dhrystones [19], Whetstones [20], Linpack [21], SPECint, and SPECfp [22][23].

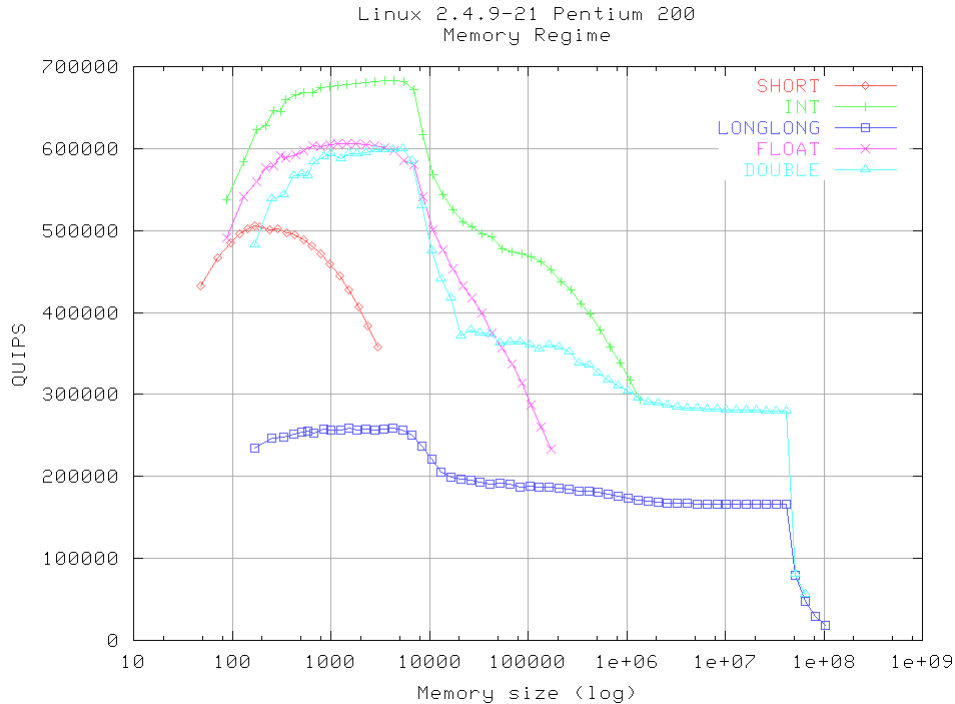


Figure 2: Hint Benchmark Results

Figure 2 shows the memory hierarchy for the various data types on the test machines. The x-axis represents the memory size in bytes and is plotted on a logarithmic (base 10) scale to reveal the full performance profile. The initial plateau represents the speed while in cache. The remaining relatively flat corresponds to the performance of computations

using main memory and the final drop off (at approximately 60Mbytes) corresponds to using virtual memory and hence accessing disk.

4.2 PCI Performance

To collect the data transfer rate from memory to LANai9 via PCI, a `gm_debug` script is used. `gm_debug` is the interface used to access debugging information in GM. By default it returns the results from the hardware benchmark of the PCI bus performed by the Myrinet adapter when the driver was loaded, along with the state of various event counters from the GM firmware, which the user might find useful.

For the 32bit/33MHz PCI bus, the theoretical PCI data rate is 132Mbytes/s. The PCI data transfer rate from the host to LANai9 is approximately 124 Mbytes/s for reading and 128 Mbytes/s for writing. This is relatively close to the theoretical transfer rate. Table 1 shows the detail of the benchmark².

DMA rate for 4096 Byte transfers (32bit/33MHz bus)	
1 st : 8 pages from bogus sdma pg, 8 to bogus rdma	bus_read (send) = 105 MBytes/s
	bus_write (recv) = 113 MBytes/s
DMA rate for 4096 Byte transfers (32bit / 33MHz bus)	
2 nd : sdma a page, rdma a page 8 times	bus_read (send) = 124 MBytes/s
	bus_write (recv) = 128 MBytes/s
DMA rate for 4096 Byte transfers (32bit / 33MHz bus)	
3 rd : sdma and rdma to/from alternating pages (coarse grain)	bus_read (send) = 124 MBytes/s
	bus_write (recv) = 128 MBytes/s
DMA rate for 4096 Byte transfers (32bit / 33MHz bus)	
4 th : sdma and rdma to/from alternating pages (fine grain)	bus_read (send) = 124 MBytes/s
	bus_write (recv) = 128 MBytes/s

Table 1: PCI Performance

The performance of these transfers is one of several limiting factors to the performance of Myrinet.

4.3 GM LogP Performance

LogP is a model for parallel algorithm design proposed by Culler et al in [24] It is intended to serve as a basis to develop fast, portable parallel algorithms and to offer guidelines to machine designers.

The LogP model is based on four parameters:

- The number P representing the number of processor-memory pairs in the machine,

² <http://www.conservativecomputer.com/myrinet/perf.html> gives a summary of the performance of various chipsets doing PCI transfers to Myrinet cards.

- The gap g representing a lower bound on the time between the transmission of successive messages, or the reception of successive messages at a processor,
- The latency L representing an upper bound on the time to transmit a message from its source processor to its destination processor, and
- The overhead O representing the amount of time for which the processor is engaged in sending or receiving a message

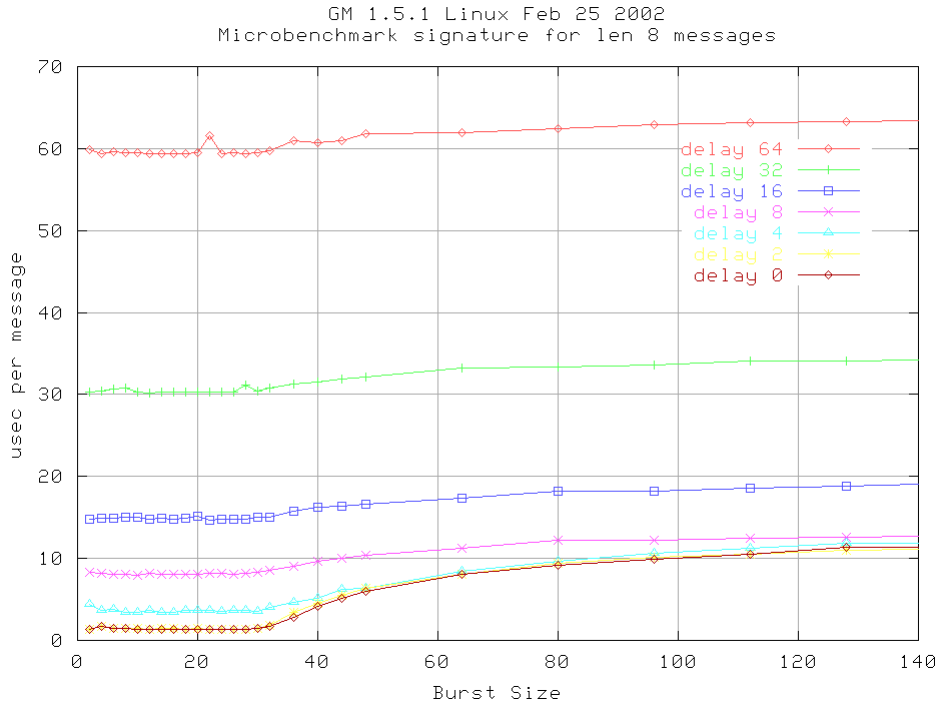


Figure 3: GM LogP delay Performance

Culler et al. in [24] used the LogP model to compare the performance of several network adaptors. Since then, the model has become a standard modelling tool for evaluating and capturing the relevant aspect of network performance.

The *logp_test* from the GM distribution implements two of the tests described in the Culler’s paper. In the first test, the sender sends a message to the receiver. Upon receipt of the message, the receiver sends it back to the sender without modifying the contents. This process is then repeated N times. This test essentially computes the Round-Trip-Time (RTT). The RTT gives the quantity of $2*(O_s + L + O_r)$. In the second test, the sender continuously sends messages to the receiver and simultaneously handles the acknowledgements from the receiver. Upon receipt of the message, the receiver sends back an acknowledgment. The average time per send is then used to create "LogP" curves from which the LogP parameters can be read.

Figure 3 shows the results from tests where a message length of 8 bytes was used to obtain the LogP parameters – g , O_s , O_r , and RTT . Messages of a given length are sent in bursts with a delay of *delta* seconds between each message. When the burst size is small, the sender can send all the messages in the burst before it receives any

acknowledgements back from the receiver. When δ is equal to zero, the average time to send the first few messages corresponds to the send overhead (O_s). We found that O_s is approximately $1.28 \mu\text{sec}$. As the burst size increases, the sender will have to handle some acknowledgements while it sends the messages in the burst. The curve transitions starts when the burst size reaches RTT/O_s . In Figure 3, the transition starts at a burst size of approximately 19. This indicates that the sender could send up to 19 messages before receiving the first acknowledgement from the receiver. As the burst size continues to increase, the curve will reach the asymptotic value of $13.2 \mu\text{secs}$. This value corresponds to g . At this stage, the sender must handle one acknowledgement for every send.

Several curves with different δ values are plotted. For $\delta < g$, the curves will approach g for large bursts. The receive overhead (O_r) can be computed by subtracting the sum of δ and O_s from g' , where $\delta > g$ and g' is the new bottleneck. In our case, O_r is approximately $-1.84 \mu\text{sec}$ with $g' = 63.40$, $\delta = 64$, $O_s = 1.28$. This implies that the receive overhead is negligible. However, it is important to note that O_r increases when the message length increases. Finally, L is computed as $RTT/2 - O_r - O_s$ and we get a latency of approximately $12.8 \mu\text{sec}$.

Figure 4 is a plot of the RTT, latency, and gap as a function of message size.

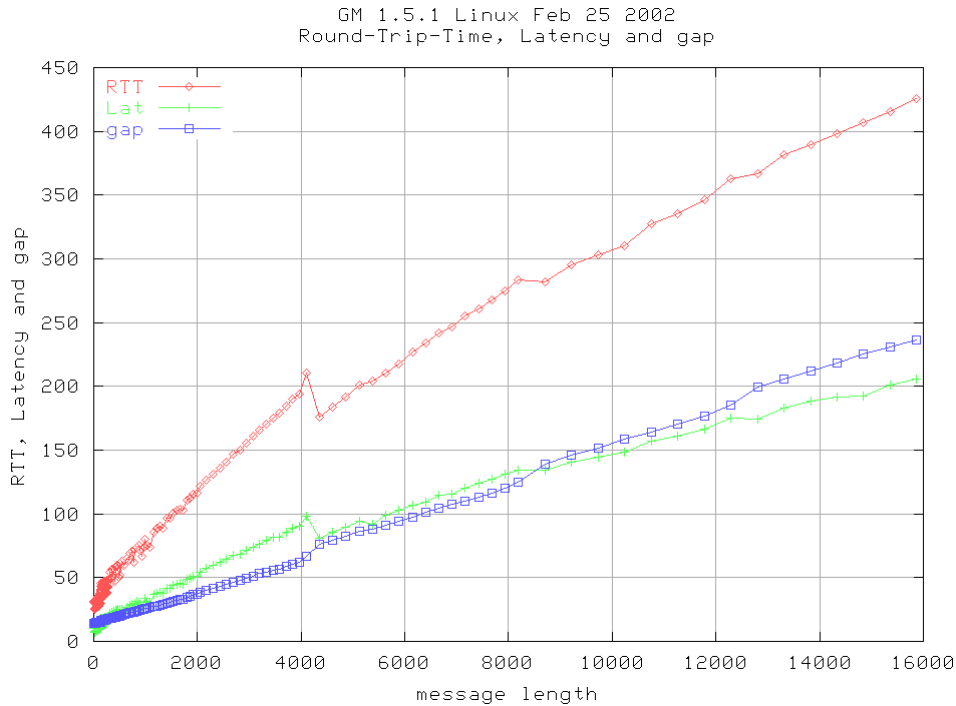


Figure 4: LogP RRT, Latency, and Gap

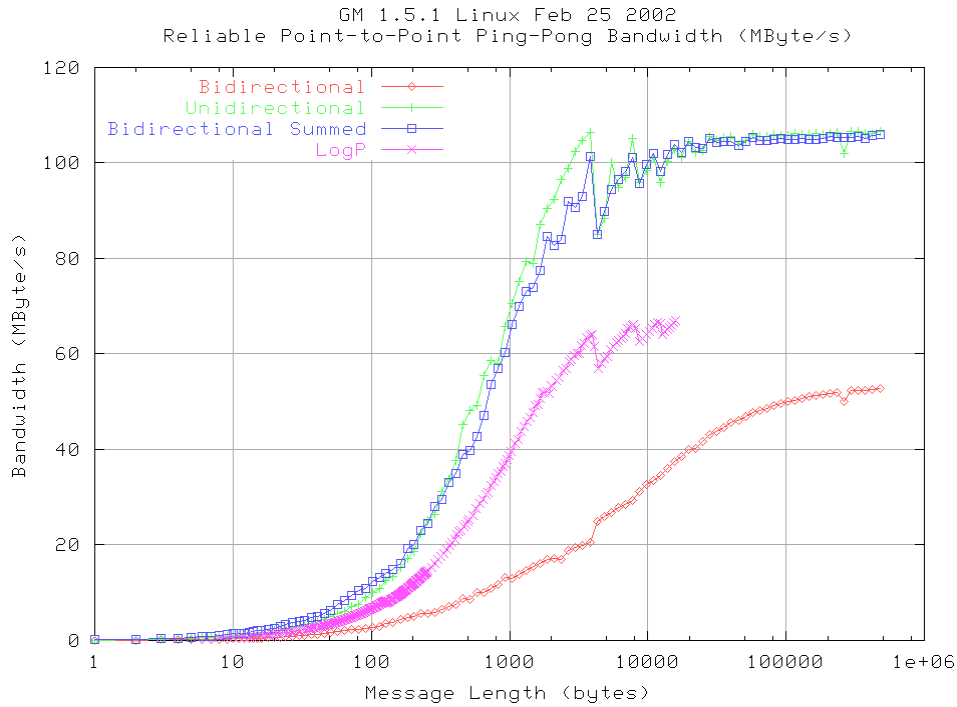


Figure 5: GM Throughput Performance

Figure 5 is a plot of the GM throughput performance. The maximum throughput obtained using the LogP model is approximately 70 Mbytes/s. The throughput is obtained by computing the value $length/g$. However, using *gm_allsize* program, the maximum attainable throughput is approximately 107 Mbytes/s for uni-directional transfer and 52 Mbytes/s for bi-directional transfer. The difference between the maximum attainable uni-directional throughput of the LogP test and *gm_allsize* test is due to the sender in the LogP test is sending and processing acknowledgments simultaneously while the sender in the *gm_allsize* test only sends messages. The maximum attainable uni-directional throughput represents 83% of the peak PCI bandwidth. From figure 5, one can also observe that the characteristic of the unidirectional curve is similar to the LogP curve with a dropout at message size approximately equals to 4 Kbytes. The reason for this dropout is that GM fragments long messages into packets of at most 4 Kbytes at the sender, and reassembles the packets into messages at the receiver. This fragmentation and reassembly is to ensure that a long message will not hog a communication channel for an extended period.

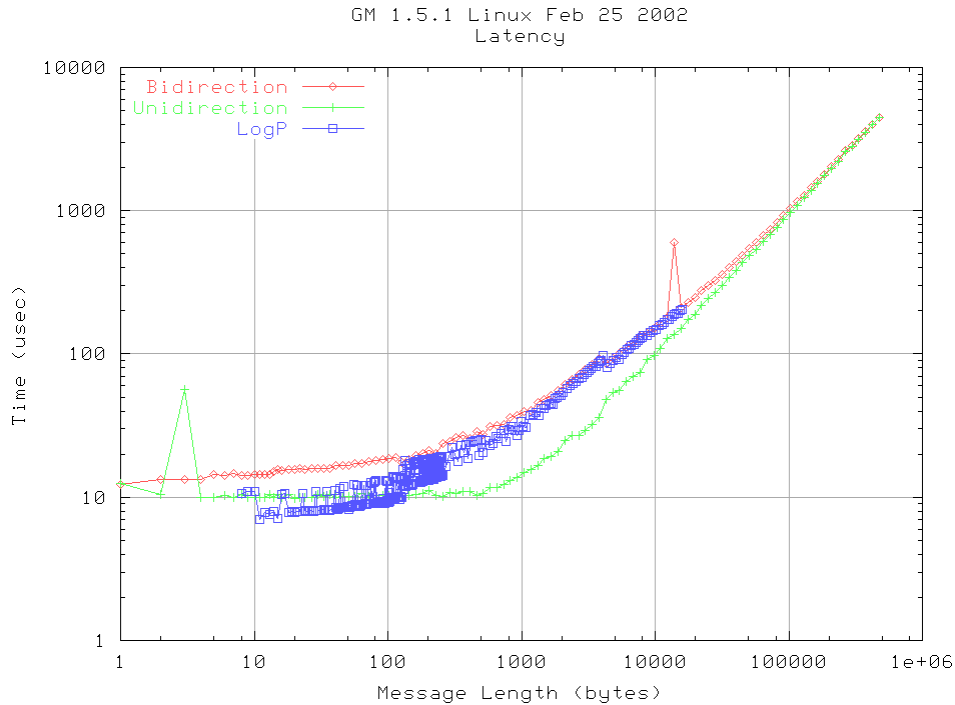


Figure 6: GM Latency Performance

Figure 6 shows the logarithm plot of GM latency. The latency of a 1-byte message for bi-directional transfer is approximately 12 μ sec and is approximately 11 μ sec for the uni-directional transfer and LogP. These latencies are composed of the time required to copy data from the host memory to the NIC via PCI/DMA, from the physical link to the receiver's end-point, and an instruction-execution component. The time required to execute the instruction-execution component approximately 6 μ sec [15] and the time to copy a one-byte message from host to the NIC is approximately 2 μ sec (see section 4.1). This implies that the latency for the link-port is roughly 3 μ sec. There is an anomaly in the plot for the uni-directional case with a message size equal to 3 bytes. However, this anomaly does not occur in a different test run.

4.4 TCP/IP over GM Performance

Myrinet packets can be of any length. This allows them to encapsulate other types of packets, such as IP packets. Since each packet is identified by type, Myrinet may carry packets of many types or protocols concurrently.

To obtain TCP/IP over GM performance, we used the GM Ethernet emulation driver, i.e., TCP/IP driver. The NetPIPE [25][26] benchmark was used to obtain the performance results. NetPIPE is a network protocol independent performance evaluation tool developed by Ames Laboratory. NetPIPE is essentially a ping-ping like program that increases the transfer block size from a single byte to large blocks until transmission time exceeds 1 second. Specifically, for each block size c , three measurements are taken for block sizes $c-p$ bytes, c bytes and $c+p$ bytes, where p is a perturbation parameter with a

default value of 3. This allows the examination of block sizes that are possibly slightly smaller or larger than an internal network buffer. Currently, NetPIPE supports TCP, PVM, and MPI communication protocols.

Figure 7 and Figure 8 show the throughput and “latency” performance of TCP/IP over GM using various socket buffer sizes. The socket buffer size determines the size of the sliding window in TCP and thus determines the number of packets that can be sent before the sender receives its first acknowledgement from the receiver. In addition, GM is configured to support “Jumbo” frames. That is the maximum transfer unit (MTU) or packet size that can be transmitted across the link is set to 9000 bytes. It well known that large MTUs can improve TCP/IP performance [3].

The maximum attainable TCP/IP performance is approximately 25 Mbytes/s using a socket buffer size of 512 Kytes. This only represents a 23% of the raw performance obtained using the pure GM protocol. This clearly illustrates the TCP/IP protocol processing is the major bottleneck.

The reason that the socket buffer size of 512 Kbytes gives the maximum TCP/IP throughput is because the maximum DMA-able memory that can be allocated in our test machines is 512Kbytes. Thus, increasing the socket buffer size would not further increase our TCP/IP performance.

The TCP/IP latency for a one-byte message is between 182 μ sec to 187 μ sec. Recall that the latency obtained from the raw GM is approximately 11 μ sec. This implies that a slightly more than 100 μ sec is spent in copying data copy from user space to system space, and from system space to the NIC memory. This further illustrates that the PCI bus is a limiting factor in communication performance.

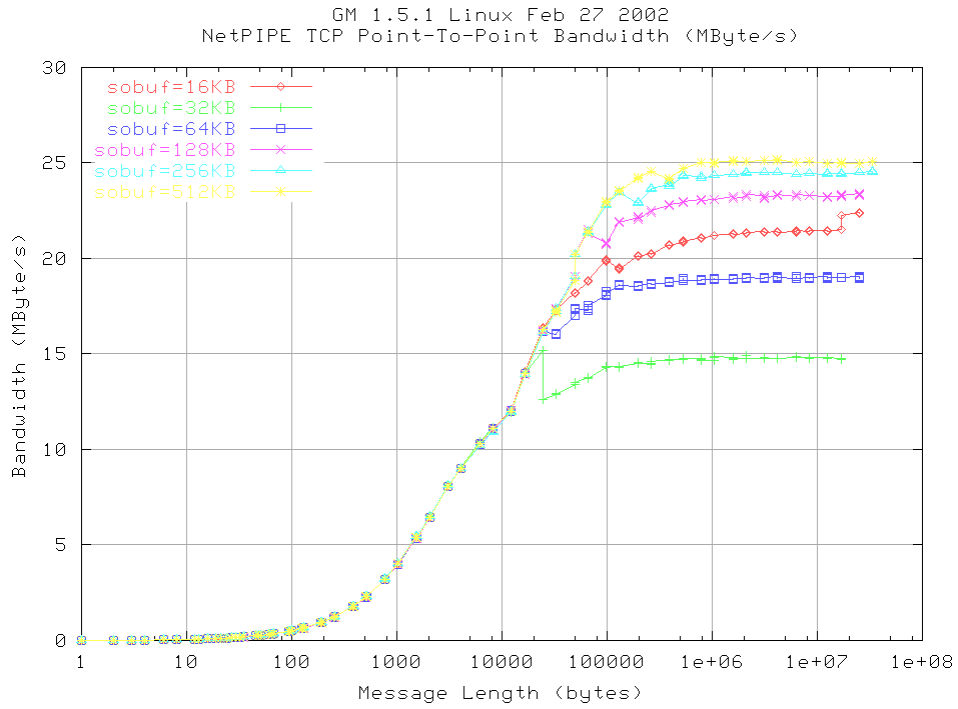


Figure 7: TCP/IP over GM Bandwidth Performance

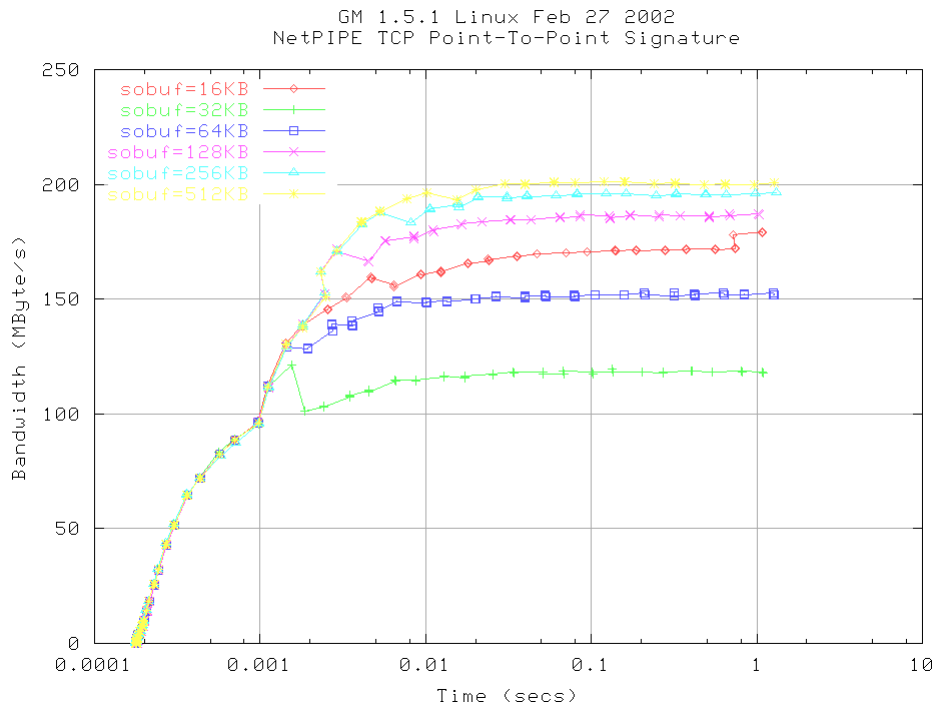


Figure 8: TCP/IP over GM Signature

4.5 MPICH over GM Performance

MPICH/GM, is based on MPICH from Argonne National Laboratories [27][28], is an implementation of MPI by Myricom. The performance results for MPICH/GM were obtained using NetPIPE benchmark.

Figure 9 and Figure 10 shows the throughput and latency performance, of MPICH/GM respectively. From Figure 9, the throughput performance is clearly being separated into two regions due to the short-long message protocol³. The pivot point is roughly 12 Kbytes. The throughput reaches a steady state of approximately 19 Mbytes/s between 4 Kbytes to 12Kbytes. Then, the throughput curve grows steeply and reaches its maximum of approximately 106 Mbytes/s for a message size of 16 Mbytes. For message sizes of greater than 16 Mbytes, the throughput curve has a severe drop because GM is accessing the virtual memory space.

The latency for a one-byte message is approximately 18 μ sec. This represents a 63% increase in time as compared to the raw latency.

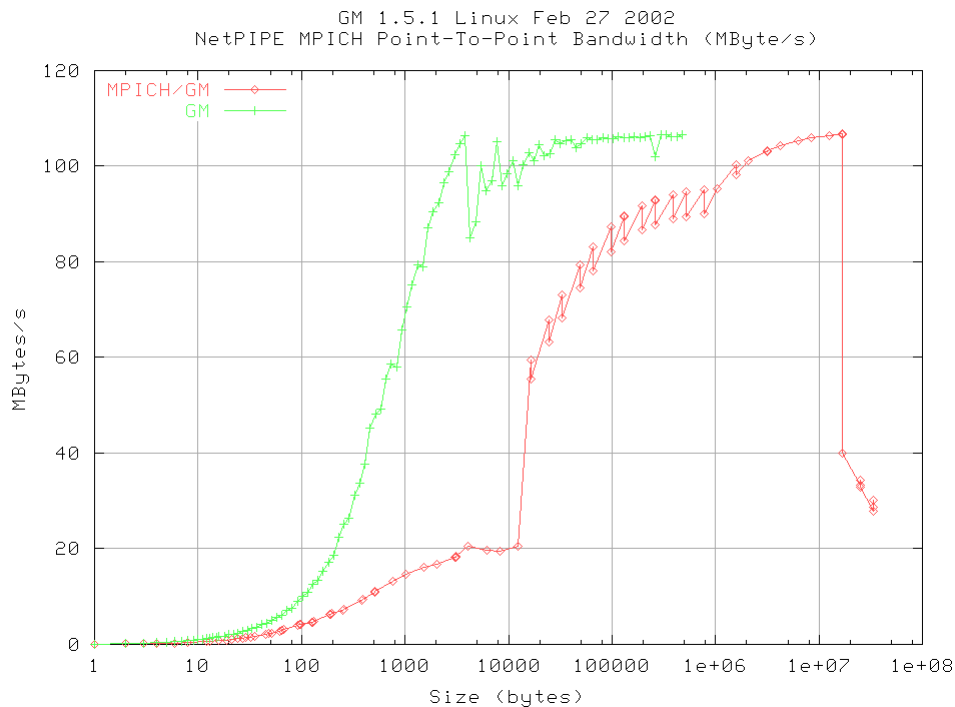


Figure 9: MPICH overGM Bandwidth Performance

³ See [28] for an explanation of MPICH short-long message protocol

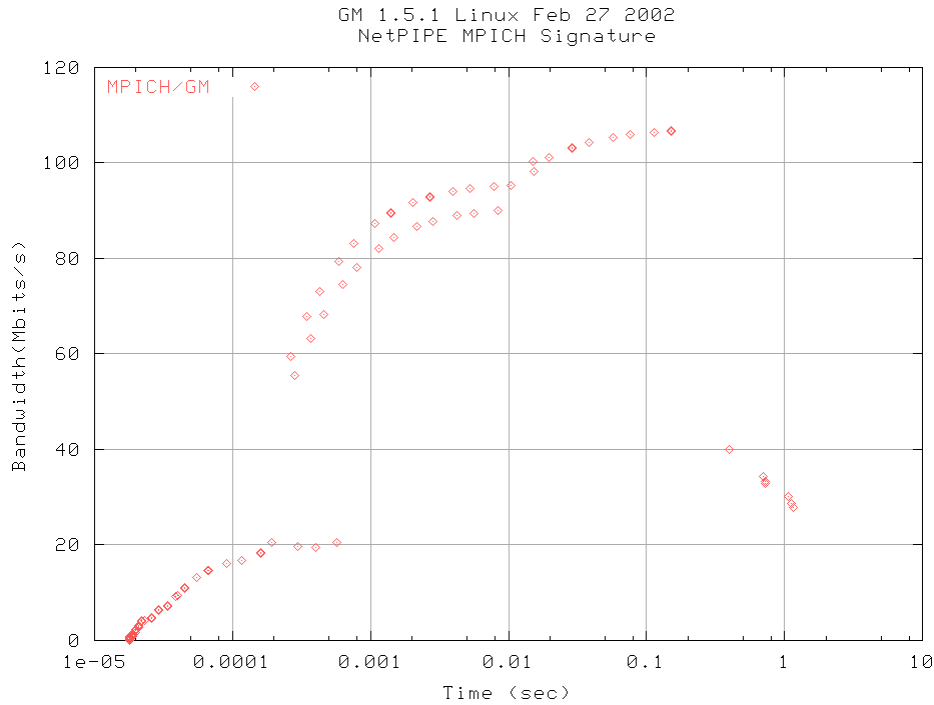


Figure 10: MPICH over GM Signature

5 Concluding Remarks

In high-speed networks with links reaching Gigabits per second, COTS-based computer systems have been pushed to their physical limits, especially where the current memory bus technology is the bottleneck for communications. This factor demands that the operating system and associated protocol software makes good use of the underlying hardware resources, such as the memory bandwidth, the CPU time and caches. However, the current generation of operating systems were initially designed and optimized for scientific computation instead of communication performance. With communications becoming a demanding aspect of computing, current operating systems are unable to handle network I/O needs. For example, the need to move a large volumes of data done through one host and passing over up to another host, and the need to touch the data to ensure its correctness, and the protocol processing needs.

Research to improve the effectiveness of an operating systems ability to handle network I/O have been going on for some time, all kinds of design schemes that integrate DMA or PIO, virtual memory techniques, buffer management, and APIs have been proposed. These design focus on the reduction or elimination of data handling overheads, including OS-by pass and zero-copy. Many of these techniques are providing promising results.

In this paper, we have presented a series of experiments. Each experiments was intended to quantify the performance characteristics of the different messaging protocols for Myrinet-2000 adaptors. Although the LogP tests measure and characterize a Myrinet

communication sub-system, it is not fully adequate as modern network interfaces exhibit characteristics (i.e. pipelining) that are difficult to capture. Moreover, the communication model of a library can make it very difficult for these benchmarks to meaningfully represent an application communication patterns. In this paper, we have used HINT, NetPIPE, and a set of tests each intended to measure the specific features of a system. These benchmarks complement the LogP micro-benchmarks and offer more insight into the impact of the communication layer an applications performance.

References

- [1] D. J. Becker, T. Sterling, D. Savarese, J. E. Dorband, U. A. Ranawak, and C. V. Packer, "Beowulf: A parallel workstation for scientific computation", *In Proceedings of the International Conference on Parallel Processing*, 1995.
- [2] Richard P. Martin, Amin M. Vahdat, David E. Culler, Thomas E. Anderson, "Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture", *ISCA 24*, 1997.
- [3] Paul A. Farrell and Hong Ong, "Communication Performance over a Gigabit Ethernet Network", *19th IEEE International Performance, Computing, and Communications Conference (IPCCC)*, 2000
- [4] T. Von Eicken, A. Basu, V. Buch, and W. Vogels, "U-NET: A User Level Network Interface for Parallel and Distributed Computing", *Proceeding of the 15th SOSP*, 1995.
- [5] S. Pakin, M. Lauria, A. Chien, "High Performance Messaging on Workstation: Illinois Fast Message (FM) for Myrinet", *Proceeding of the International Conference on Supercomputing'95*, 1995
- [6] T. Von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauer, "Active Messages: A Mechanism for Integrated Communication and Computation", *International Symposium on Computer Architecture*, 1992
- [7] Myricom Inc., "The GM Message Passing System", 2000
- [8] Intel Corporation, "Virtual Interface Architecture Specification version 1.0", 1997. <http://developer.intel.com/design/servers/vi/>
- [9] E. Speight, H. Abdel-Shafi, J. K. Bennett, "Realizing the Performance Potential of the Virtual Interface Architecture", *Proceedings of the International Conference on Supercomputing'99*, 1999.
- [10] InfiniBand Trade Association. <http://www.infinibandta.org>
- [11] InfiniBand Trade Association. InfiniBand Architecture Specification, release 1.0, October 2000. <http://www.infinibandta.org>
- [12] J. Smith and C. Brendan S. Traw. "Giving Applications Access to Gb/s Networking," *IEEE Network*, July 1993.
- [13] Darrell Anderson, Jeffrey S. Chase, Syam Gadde, Andrew J. Gallatin, Kenneth G. Yocum, and Michael J. Feeley, "Cheating the I/O bottleneck: Network storage with Trapeze/Myrinet", *Proceedings of the 1998 Usenix Technical Conference*, June 1998.
- [14] N. Boden, D. Cohen, and R. Felderman, "Myrinet: a gigabit per second local-area network", *IEEE Micro 14(1) 29*, February 1994.
- [15] Myricom Inc., "Guide to Myrinet-2000 Switches and Switch Network", revision 27, 2001
- [16] Myricom Inc., "Guide to Myrinet/PCI Host Interface", 2002
- [17] Gustafson, John L. and Snell, Quinn O., "HINT: A New Way To Measure Computer Performance", *In Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences*, Wailela, Maui, HI, January 1995.
- [18] HINT, <http://www.scl.ameslab.gov/HINT>.

- [19] R. Weicker, "Dhrystone: A Synthetic Systems Programming Benchmark," *Communications of the ACM*, Volume 27, Number 10, October 1984.
- [20] H.J. Curnow, and B. A. Wichmann, "A Synthetic Benchmark", *Computer Journal*, 19,1, February 1976J.
- [21] J. Dongarra, "Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment," <http://www.netlib.org/benchmark>, 2002.
- [22] J. Dongarra and W. Gentzsch, eds, "Computer Benchmarks", North Holland, Amsterdam, 1993
- [23] The SPEC Web site is <http://www.specbench.org/osg/cpu95>
- [24] D. E. Culler, R. Karp, D. A. Patterson, A. Sahay, K. E. Schauer, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a realistic model of parallel computation, *Proceedings. Of the 4th SIGPLAN Symp. On Principles and Practices of Parallel Programming*, ACM, May 1993.
- [25] Q.O. Snell, A.Mikler, and J.L. Gustafson, "NetPIPE: A Network Protocol Independent Performance Evaluator", *In IASTED International Conference on Intelligent Information Management and Systems*, June 1996.
- [26] NetPIPE, <http://www.scl.ameslab.gov/netpipe>.
- [27] W.D. Gropp and E. Lusk and N. Doss and A. Skjellum, "A high-performance, portable implementation of the (MPI) message passing interface standard", *Parallel Comp.* 22, (1996).
- [28] W.D. Gropp and E. Lusk, "User's Guide for MPICH, a Portable Implementation of MPI", Argonne National Laboratory (1996), ANL-96/6.