Messaging on Gigabit Ethernet: Some experiments with GAMMA and other systems

Giuseppe Ciaccio
DISI, Università di Genova
via Dodecaneso 35, 16146 Genova, Italy
ciaccio@disi.unige.it

Abstract

The Genoa Active Message MAchine (GAMMA) is a lightweight communication system based on the Active Ports paradigm, originally designed for efficient implementation over low-cost Fast Ethernet interconnects.

In this paper we report about the recently completed porting of GAMMA to the Packet Engines GNIC-II and the Netgear GA620 Gigabit Ethernet adapters, and provide a comparison among GAMMA, MPI/GAMMA, TCP/IP, and MPICH, on such high-performance commodity adapters, using different performance metrics. With a combination of low end-to-end latency (9.5 μs with GNIC-II, 32 μs with GA620) and high transmission throughput (more than 96 MByte/s with GNIC-II, more than 103 MByte/s with GA620, the latter obtained without changing the firmware of the adapter), GAMMA demonstrates the potential for Gigabit Ethernet lightweight protocols to yield messaging performance comparable to the best Myrinet protocols. This result is of interest, given the envisaged drop in cost of Gigabit Ethernet due to the forthcoming transition from fiber optic to UTP cabling and ever increasing mass market production of such standard interconnect.

We also reposts about a technique for message fragmentation that is commonly exploited to increase the throughput with short message. When a different, though more widely used, performance metrics is considered, such a technique results into a performance loss rather than improvement.

1. Introduction

The low-cost processing power of clusters of Personal Computers (PCs) can be easily exploited by means of high-level, industry-standard Application Programming Interfaces (APIs), MPI being the most important of them.

Several open-source implementations of MPI (e.g., MPICH) can run on Linux-based Beowulf-type clusters on top of standard TCP/IP sockets. However it is well known that parallel jobs characterized by medium/fine grain parallelism exchange messages of small size (few KBytes) among processors, and that general-purpose protocols such as TCP/IP make very inefficient use of the interconnect for short messages.

On Fast Ethernet, a lightweight messaging system like the Genoa Active Message MAchine (GAMMA) [5] provides far better performance at no additional hardware cost compared to the Linux TCP/IP stack. The GAMMA project started in 1996 and was targeted right from the beginning to low-cost Fast Ethernet interconnects. GAMMA implements a non-standard communication abstraction called *Active Ports* [8], derived from Active Messages, and provides best-effort as well as flow-controlled communication routines. With an end-to-end latency ranging between 12 and 20 μs, depending on the hardware configuration and the Network Interface Card (NIC) used, GAMMA provides adequate communication and synchronization primitives for tightly coupled, fine-grain parallel applications on inexpensive clusters. A complete yet efficient implementation of MPI atop GAMMA, based on a port of MPICH, is available [7, 9].

The link speed offered by Fast Ethernet is insufficient for many communication intensive parallel jobs to scale up. This justifies the interest towards the two most famous Gigabit-per-second interconnects, namely, Myrinet and the more recent Gigabit Ethernet.

The inefficiency of TCP/IP is exacerbated with Gigabit-class networks, since the physical transmission time becomes negligible compared to the time spent in the traversal of the protocol stack. General-purpose implementations of TCP/IP cannot run efficiently on Gigabit-per-second links, no matter how long data streams and how fast host CPUs are (people exhibiting brilliant performance numbers with TCP/IP sockets on Gigabit Ethernet actually must set a very large socket buffer size, in the order of a megabyte, which is not really a scalable solution). Thus, lightweight messaging systems become a key ingredient for an effective use of

Gigabit-class networks.

The higher per-port cost of Gigabit Ethernet compared to Myrinet was due to the fiber-optic cabling of the former. However, a substantial drop in cost of Gigabit Ethernet can be envisaged in the near future. This will eventually push Gigabit Ethernet into a much larger segment of the marketplace, characterized by competition among different vendors and a potentially very large base of installation; eventually, the per-port cost of Gigabit Ethernet will become negligible compared to the cost of a single PC, in much the same way as it occurred with Fast Ethernet. In our opinion, Myrinet will never enjoy such a large diffusion: its marketplace segment (system-area networks) will remain narrow compared to LANs, and therefore its price will never drop so much.

Moreover, some Gigabit Ethernet NICs are programmable in much the same way as Myrinet is. For instance, the NetGear GA620 NIC, a cheap (less than 300 US dollars at the time of writing) clone of the Alteon AceNIC adapter, comes with two on-board microprocessors and 512 KBytes of on-board RAM, and can run a possibly self-made custom firmware (we should call it "software" and not "firmware", due to its volatility) to support efficient, low overhead messaging.

An important difference between Myrinet and Gigabit Ethernet concerns message fragmentation. Myrinet does not pose any limitations on the size of data transfers at the physical level, and fragmentation of messages into packets is only aimed at pipelining long data transfers across the communication path [13]. However, Gigabit Ethernet is a packet-oriented network fabric. The IEEE 802.3 standard poses the limit of 1526 bytes for the frame size, regardless of the link speed, which implies a non-negligible overhead at Gigabit-per-second speed. But the GA620 NIC also supports so called "jumbo frames", that is, packets whose size exceeds the limit imposed by the standard. By allowing a packet size of up to 9000 bytes, this NIC provides a better exploitation of the physical link and helps decrease the message fragmentation overhead, with a positive impact on communication efficiency with long data transfers.

Another difference between Myrinet and Gigabit Ethernet is concerned with the reliability of the physical medium, especially in case of network congestion which may cause packet losses. Myrinet prevents congestion by using hardware mechanisms for back-pressure, whose practical effectiveness has been demonstrated. Gigabit Ethernet uses hardware-exchanged control packets to block senders in case of congestion hazard, according to the IEEE 802.3x specification; this should in principle avoid congestion, and packet losses thereof, although nobody could assess the effectiveness of this mechanism so far.

A difference of secondary concern is the higher communication latency of current Gigabit Ethernet switches (in the order of 3 to 4 μs), compared to the very low latency of a Myrinet switch (less than 1 μs).

To sum up, in our opinion Gigabit Ethernet is a promisingly successful and cheaper alternative to Myrinet under any respect, and an efficient lightweight protocol is definitely a must to make best use of this technology, although it is likely that Myrinet will continue to improve in order to keep higher performance levels compared to state-of-art commodity interconnect fabrics.

In order to prepare for the transition to inexpensive Gigabit Ethernet, we started developing a prototype of GAMMA for the Packet Engines GNIC-II Gigabit Ethernet adapter. This device was one of the first Gigabit Ethernet NICs to be shipped. It closely resembled most Fast Ethernet NICs as for internal architecture. It was not programmable, nor did it support "jumbo frames". GAMMA for GNIC-II was ready in September 1999; although Packet Engines discontinued the NIC a few months later, that first prototype of GAMMA was indeed useful to experimentally demonstrate the feasibility and success of lightweight communication protocols on next-generation inexpensive LANs.

As a further step, we engaged ourselves in implementing GAMMA atop the Netgear GA620 Gigabit Ethernet NIC. Although this NIC could even be reprogrammed from scratch in order to run whatever support to fast communication, this implementation of GAMMA does not require any changes to the original NIC firmware.

2. Outline and rationale of the architecture of GAMMA

A NIC is an interface between a host CPU and a network; as such, each NIC must implement suitable mechanisms to cooperate with the host computer, on one hand, and the network, on the other hand. A modern NIC cooperates with the host computer using a data transfer mode called *Descriptor-based DMA* (DBDMA). With the DBDMA mode, the NIC is able to autonomously set up and start DMA data transfers. To do so, the NIC scans two pre-computed and static circular lists called *rings*, one for transmit and one for receive, both stored in host memory. Each entry of a ring is called a *DMA descriptor*.

A DMA descriptor in the transmit ring contains a pointer (a physical address) to a host memory region containing a *fragment* of an outgoing packet; therefore, an entire packet can be specified by chaining one or more send DMA descriptors, a feature called "gather".

Similar to a descriptor in the transmit ring, a DMA descriptor in the receive ring contains a pointer (a physical address, again) to a host memory region where an incoming packet could be stored. The analogous of the "gather" feature of the transmit ring is here called "scatter": more descriptors can be chained to specify a sequence of distinct

memory areas, and an incoming packet could be scattered among them.

A NIC operating in DBDMA mode allows a greater degree of parallelism in the communication path, according to a producer/consumer behaviour. At the transmit side, while the NIC "consumes" DMA descriptors from the transmit ring and operates the host-to-NIC data transfers specified in the descriptors themselves, the CPU runs the protocol stack and "produces" the necessary DMA descriptors for subsequent data transfers. The reverse occurs at the receive side. Since both sides are decoupled from each other, the communication path works like a pipeline whenever traveled by a sequence of data packets, with a potentially high throughput.

At the sender side, a proper organization of the send routine could avoid a memory copy by exploiting the "gather" feature: the header could be pre-computed and stored somewhere in kernel space, the payload could be pointed to directly in user space, and the NIC would autonomously arrange them contiguous into its on-board transmit FIFO and send the whole packet. This would imply a non-blocking semantics of the send routine. Blocking semantics could then be easily enforced, by putting a busy-waiting on transmission completion at the end of the non-blocking send; this way we would have a zero-copy blocking send whose overhead would however be much higher compared to a more classical "one-copy" blocking send, since the latter would terminate as soon as user data had travelled the memory bus back and forth to be copied to a temporary buffer, and usually the memory bus has much higher a transfer rate compared to the I/O bus. Thus, a convenient messaging system should provide zero-copy non-blocking send routines (using the "gather" feature of the NIC) together with "one-copy" blocking send routines.

At the receiver side, however, there is no way to avoid a memory copy if non-programmable NICs are leveraged. Indeed, the final destination in user space for the payload of an incoming packet can be determined only *after* inspecting the header, which implies the packet be already stored somewhere, namely, into a temporary buffer. The only way to avoid a memory copy at the receiver side is to run the (header-processing part of the) communication protocol by the NIC itself, so as to let it inspect the headers when packets are still in its on-board receive FIFO. Another solution uses the "page remapping" trick, but is limited to page-aligned user buffers [10].

The GAMMA communication protocol is implemented according to the above analysis, and thus minimizes the CPU involvement on send in the blocking as well as the non-blocking case, with no data copies on send in the latter; whereas the CPU involvement on receive is always proportional to the size of the arriving messages, due to the unavoidable data copy on receive. Both best-effort and flow-controlled communications are provided, the latter based on a simple (and static, for the moment) end-to-end credit algorithm.

In the current version of GAMMA, no packet retransmission is carried out on (very rare) communication errors: the long-term reliability at application level is demanded to suitable checkpointing policies of the user job. Network congestion (which is the main source of packet losses) is avoided by properly tuning the end-to-end flow control, in case the LAN devices do not support the IEEE 802.3x congestion control mechanisms.

The GAMMA protocol is independent of the specific network adapter, but is based on a small set of NIC-dependent low-level basic operations (for producing/consuming data packets, enabling/disabling IRQs, reading the device status), that we call the *Basic Interface for Network Device Drivers* (BIND^2). Porting GAMMA to a new NIC requires writing the BIND^2 layer for that NIC, and modifying the Linux device driver of that NIC in order to "graft" the GAMMA protocol in the driver itself. The resulting messaging system is thus implemented at *kernel level* in the OS architecture, and allows IP-based network services to coexist with GAMMA on the same LAN.

3. Gigabit Ethernet: performance results and comparisons

Two different measurement platforms were used for the tests. The first cluster was a pair of PCs with Pentium II 450 MHz CPU, ASUS motherboard (Intel 440 BX chipset), 100 MHz SDRAM, and Packet Engines GNIC-II NIC. The second cluster was a pair of PCs with AMD Athlon 500 MHz CPU, Microstar K7 Pro motherboard (AMD 751/756 chipset), 100 MHz SDRAM, and Netgear GA620 NIC. In both cases the interconnection was back-to-back by a fiber-optic cable.

Self-made microbenchmarks have been run to measure the average communication performance of GAMMA, MPI/GAMMA, Linux 2.2.13 TCP/IP sockets, and MPICH atop TCP/IP sockets (P4 channel device).

With the first platform, it was also temporarily possible to connect the two machines by an Intel Express Gigabit switch; the latency increased by 3.6 μs on average and the impact on communication throughput was negligible.

3.1. End-to-end latency and throughput

The end-to-end communication performance has been measured by usual "ping-pong" microbenchmarks, which measure the average round-trip time for a message of fixed size S and divide it by two. The *end-to-end latency* is obtained by setting S equal to the minimum message size allowed by the messaging system (zero with GAMMA and

NIC (host CPU)	GAMMA	MPI/ GAMMA	TCP/ IP	MPICH
Packet Engines GNIC-II (Pentium II 450)	9.5	12.1	132	321
NetGear GA620 (Athlon 500)	32	35	65	112

Table 1. End-to-end latency (μs) of GAMMA, MPI/GAMMA, Linux 2.2.13 TCP/IP, and MPICH atop TCP/IP, with the GNIC-II and GA620 adapters.

GAMMA/ GNIC-II	BIP	FM	PM	GM
13.1	4.3	9	7.5	9 – 21

Table 2. End-to-end latency (μs) of GAMMA on GNIC-II (Gigabit Ethernet, switch included), compared to BIP, FM, PM, and GM (Myrinet).

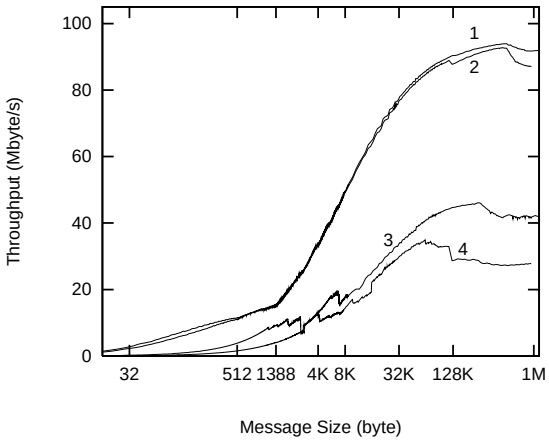

Figure 1. End-to-end throughput of GAMMA (curve 1), MPI/GAMMA (curve 2), Linux 2.2.13 TCP/IP (curve 3), and MPICH atop TCP/IP (curve 4), using the GNIC-II adapter.

MPI, one with TCP sockets). Large values of S allow to estimate the *end-to-end throughput*, not to be confused with the *transmission throughput* as measured by another class of well known microbenchmarks, namely, the "unidirectional stream" tests (Section 3.2).

Latency numbers from GAMMA, MPI/GAMMA, Linux 2.2.13 TCP/IP and MPICH are reported in Table 1.

The impressively low latency achieved by GAMMA with the GNIC-II adapter accounts for the efficiency of the protocol in itself, which indeed is able to deliver much of the raw performance of the interconnect to the upper layers. Even taking into account the hardware latency of a Gigabit Ethernet switch (in the order of 3 to 4 μs on average), the end-to-end latency remains at very low levels indeed. On the other hand, the not brilliant result achieved by the same protocol on a faster PC with the GA620 adapter shows that the GA620 NIC is "slower" indeed, compared to the GNIC-II.

A latency comparison between GAMMA/GNIC-II (inclusive of switch latency) and the main Myrinet-based messaging systems (BIP [13], FM [11], PM [16], and GM [12]) is reported in Table 2. The table shows that Gigabit Ethernet is able, at least in principle, to compete with Myrinet as for end-to-end latency.

End-to-end throughput curves for the GNIC-II and GA620 Gigabit Ethernet adapters are depicted in Figures 1 and 2, respectively.

In Figure 1, the slight decrease in throughput of GAMMA (curve 1) and MPI/GAMMA (curve 2) at around 512 KBytes with the GNIC-II adapter is due to the overflow of L2 cache, which exposes the quite poor bandwidth of the CPU-to-RAM bus in the old-fashioned PCs used for that experiment (Pentium II 450 MHz). This could happen because GAMMA performs a memory-to-memory copy on receive. However, more recent PCs have better CPU-to-RAM buses; the impact of the memory-to-memory copy has thus disappeared from the throughput curve (Figure 2).

Again from Figure 1, it is evident that stacking MPICH atop GAMMA had only a marginal impact on communication performance. The same does not hold with plain MPICH atop TCP/IP sockets. The reason for this, is that the porting of MPICH atop GAMMA has been made at the ADI level, and thus the resulting MPI/GAMMA stack is thin compared to the standard MPICH/P4/TCP stack. Needless to say, the efficiency of MPICH atop Linux TCP/IP with short messages clearly remains too low for running a tightly coupled, fine-grain parallel job on a commodity Gigabit Ethernet cluster.

By comparing the TCP/IP curves in Figures 1 and 2, it emerges that the efficiency of Linux TCP/IP with short messages changes significantly from NIC to NIC. This in our opinion depends on the quality of the device driver. The Linux driver supporting the Alteon TIGON-II Gigabit Ethernet chipset (the one equipping a number of Gigabit Ethernet NICs, including the GA620) is much more optimized compared to the Linux driver for the GNIC-II adapter.

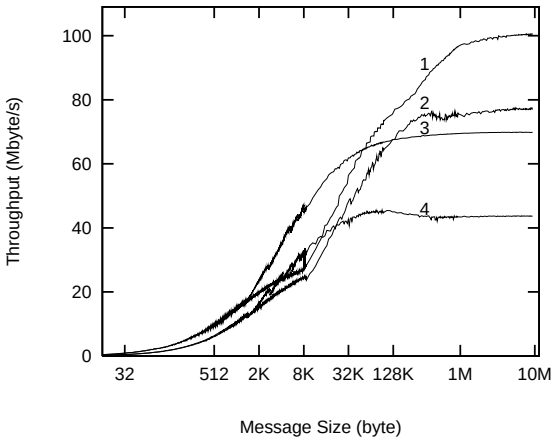

Figure 2. End-to-end throughput of GAMMA with "jumbo" frames (curve 1), Linux 2.2.13 TCP/IP with "jumbo" frames (curve 2), GAMMA with normal frames (curve 3), and Linux 2.2.13 TCP/IP with normal frames (curve 4), using the GA620 adapter.

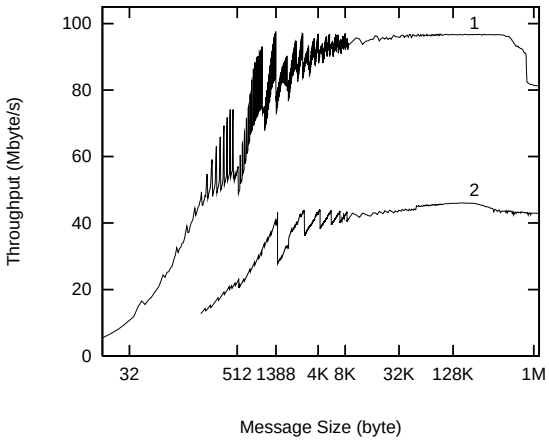

Figure 3. Transmission throughput of GAMMA (curve 1) and Linux 2.2.13 TCP/IP (curve 2) using the GNIC-II adapter.

GAMMA/ GA620	BIP	FM	PM	GM (32 bit PCI)
103.5	126	92	113.5	100

Table 3. Peak throughput (MByte/s) of GAMMA on GA620 (Gigabit Ethernet, "jumbo frames" enabled), compared to BIP, FM, PM, and GM (Myrinet).

3.2. Transmission throughput

We call *transmission throughput* the communication throughput as measured by the well known "unidirectional stream" test. This technique requires a transmitter process to send a long data stream to a receiver process; the stream sizes N bytes total, and consists of a sequence of messages, S bytes each (we suppose N be multiple of S). The sender measures the average time $T(S)$ needed to transmit a single such stream at maximum speed allowed by the messaging system, then computes the transmission throughput as $N/T(S)$, which leads to a function of message size S. Flow control must be used, or implemented by the microbenchmark if not available, to ensure that the transmission speed be sustained by the receiver. In our experiments, N was set to 10 MBytes.

Transmission throughput curves of GAMMA and Linux 2.2.13 TCP/IP for the GNIC-II and GA620 Gigabit Ethernet adapters are depicted in Figures 3 and 4, respectively.

The transmission throughput of GAMMA on the GNIC-II adapter (Figure 3, curve 1) shows a severe decrease at around 512 KBytes. Similar to what happens with the end-to-end throughput, this is due to the overflow of L2 cache in the memory-to-memory copy performed by GAMMA on receive, which exposes the quite poor bandwidth of the CPU-to-RAM bus in the old-fashioned PCs used for that experiment (Pentium II 450 MHz). This no longer occurs with more recent PCs (Figure 4).

In all cases it can be noted that the transmission throughput curves are much steeper than the corresponding end-to-end throughput curves. Indeed, by no means can the two kinds of metrics be compared with each other; nonetheless, some authors use to mix them when comparing performance of different messaging systems (like in, e.g., [4], where the transmission throughput of Berkeley VIA on Myrinet is compared to the end-to-end throughput of BIP).

A throughput comparison between GAMMA and the main Myrinet-based messaging systems (BIP [13], FM [11], PM [16], and GM [12]) is not straightforward, because in [13] only end-to-end throughput is reported, while in [11] and [12] the microbenchmarks used for the measurement are not clearly described, and in [16] only transmission throughput is reported (apparently). Nevertheless, in Table 3 we show a comparison among the various "peak throughput" numbers claimed in the aforementioned papers, based on the fact that *peak* end-to-end and transmission throughput numbers usually coincide. The table shows that Gigabit Ethernet is indeed able to compete with Myrinet in terms of peak throughput, provided a good lightweight protocol (GAMMA in our case) be available.

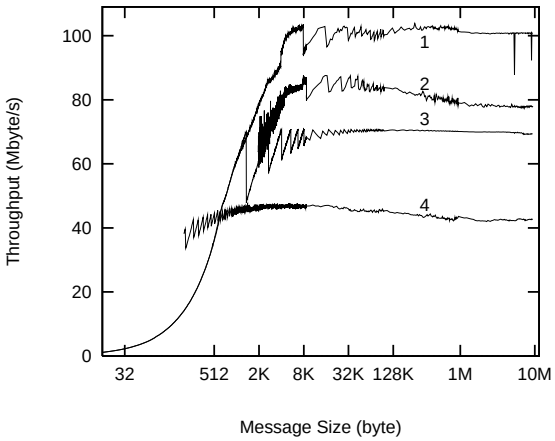

Figure 4. Transmission throughput of GAMMA with "jumbo" frames (curve 1), Linux 2.2.13 TCP/IP with "jumbo" frames (curve 2), GAMMA with normal frames (curve 3), and Linux 2.2.13 TCP/IP with normal frames (curve 4), using the GA620 adapter.

3.3. Pros and cons of different message fragmentation techniques

Fragmenting a message into packets is a need when the total message size exceeds the maximum MTU size of the network. Fragmentation increases the CPU overhead for header processing, and leads to a lower utilization rate of the physical link. However, since the end-to-end communication path is a pipeline, fragmentation might also leads to a much better end-to-end throughput, thanks to the exploitation of the parallelism among pipe stages in the communication path. For this reason, message fragmentation has been exploited also with Myrinet, which is not packet oriented [13, 11]. Even with Fast Ethernet, it can be convenient to fragment a message even if smaller than the maximum allowed MTU size [6].

To improve the end-to-end throughput with short messages, GAMMA can fragment each short message into packets whose size is not necessarily maximal. Of course, this increases the number of packets exchanged. The optimal fragment size depends on the total message size and also depends on the performance profile of the communication hardware.

We have implemented such a fragmentation technique in the GAMMA driver for the GNIC-II adapters. Due to the lack of a satisfactory performance model of the whole communication system, we could only find an empirical formula for optimal fragmentation valid only for the GNIC-II adapter (see Table 4).

The positive impact of this "adaptive" fragmentation technique on the GAMMA *end-to-end* throughput curve is clearly shown in Figure 5. For instance, a 64% improvement in throughput is obtained with 1388 byte messages.

However, the very same fragmentation technique has a negative impact on the GAMMA *transmission* throughput, as clearly apparent from Figure 6. This is due to the increased number of exchanged packets, with a consequent higher transmission overhead (at the NIC level).

As a general conclusion, we argue that changing the fragmentation policy not always yields a performance improvement; indeed, it may depend on which performance metrics the user is concerned with, as well as the per-packet overhead incurred by the various components of the messaging system.

Message size (bytes)	Packet size (bytes)
0 to 512	128
513 to 1664	256
1665 to 5148	384
5149 to 11000	768
11001 to 12000	896
$>$ 12000	1408

Table 4. Optimal size of fragments as an empirical function of the message size (GNIC-II network adapters); the optimization is w.r.t. end-to-end throughput.

4. Discussion of the results

We briefly sketch the main conclusions that can be drawn from the various performance figures presented so far:

- The communication performance of GAMMA on Gigabit Ethernet is comparable to the one of many lightweight protocols running on Myrinet, even taking into account the additional latency of a Gigabit Ethernet switch (less than 4 μs). However, the observed performance differs from NIC to NIC. On the GA620 adapters, GAMMA could yield more than 103 MByte/s peak throughput but only using "jumbo frames", whereas it saturates at 71 MByte/s with standard frames, and in both cases the end-to-end latency is quite high (32 μs). On the other hand, GAMMA on Packet Engines GNIC-II adapters raises a very high throughput (97 MByte/s) with no need of "jumbo frames", and the end-to-end latency appears to be impressively low (9.5 μs).
- The performance degradation caused by stacking MPI atop GAMMA is very modest; the same does not hold with MPI atop TCP/IP sockets. This is because the

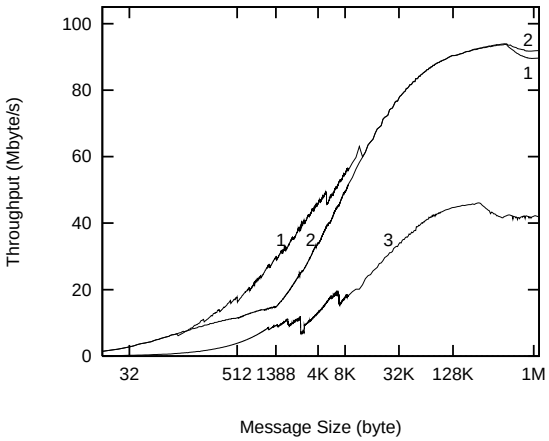

Figure 5. Positive impact of adaptive fragmentation on end-to-end throughput of GAMMA (GNIC-II adapter). Curve 1 is obtained with the adaptive fragmentation, curve 2 is obtained with standard fragmentation. End-to-end throughput of Linux 2.2.13 TCP/IP is reported as a reference (curve 3).

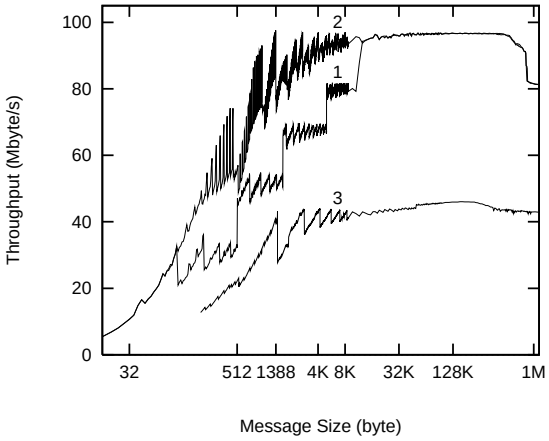

Figure 6. Negative impact of adaptive fragmentation on transmission throughput of GAMMA (GNIC-II adapter). Curve 1 is obtained with the adaptive fragmentation, curve 2 is obtained with standard fragmentation. Transmission throughput of Linux 2.2.13 TCP/IP is reported as a reference (curve 3).

porting of MPICH atop GAMMA was made at the ADI level, and thus the resulting MPI/GAMMA stack is thin compared to the standard MPICH/P4/TCP stack.

- The efficiency of Linux TCP/IP with short messages greatly depends on the device driver used. The Linux driver supporting the Alteon TIGON-II Gigabit Ethernet chipset (the one equipping a number of Gigabit Ethernet NICs, including the GA620) is much more optimized compared to the Linux driver for the GNIC-II adapter.

- The efficiency of MPICH atop Linux TCP/IP with short messages remains too low for running a tightly coupled, fine-grain parallel job on a commodity Gigabit Ethernet cluster.

- Linux TCP/IP is not able to saturate Gigabit Ethernet: its peak throughput is 87 MByte/s with the GA620 adapters using "jumbo frames", and only 42 MByte/s with the GNIC-II adapters using standard frames. The scenario is completely different over Fast Ethernet, where Linux TCP/IP is indeed able to almost completely saturate the physical link, thanks to recent improvements with device drivers and protocol.

- Changing the message fragmentation technique not always yields a performance improvement. The use of an adaptive fragmentation technique which takes the total message size into account can significantly improve the *end-to-end* throughput with short messages, because of a better exploitation of the pipeline structure of the communication path. On the other hand, the same technique also leads to a degradation of the *transmission* throughput, because of an increase of the total send overhead due to the larger number of exchanged packets.

5. Related work on Gigabit Ethernet

Not much research work has been concerned with efficient messaging over Gigabit Ethernet clusters so far; indeed, most recent and ongoing research efforts on Gigabit-class cluster interconnects have focused on Myrinet, mainly because it offered device programmability and very good link performance since quite a long (six years) time ago.

M-VIA [1] is a user-level software emulation of the Virtual Interface Architecture [2], a very low-level API for inter-process communication. M-VIA has been implemented on a number of network adapters, including the Packet Engines GNIC-II Gigabit Ethernet NIC; in this case, its communication performance (19 μs latency and 60 MByte/s peak throughput) is not that brilliant, in spite of user-level communication architecture, low API abstraction

level (messages larger than the MTU size of the network are not allowed, given that no message fragmentation/reassembly is provided), and best-effort quality of service (no flow control).

MESH [3] is another user-level messaging system, with abstraction level comparable to VIA (that is, message size cannot exceed the MTU size of the network) and best-effort quality of service (no flow control). It has been implemented for the Intel EtherExpress Pro 100 Fast Ethernet NIC, and the Alteon AceNIC Gigabit Ethernet adapter; with the latter device, MESH reported really great communication performance (24.5 μs latency and 115 MByte/s *transmission* throughput using "jumbo frames"). MESH claims a very low overhead incurred by the CPU during communication; however, such low overhead was evaluated by means of unrealistic "dummy" ping-pong tests, where user data were assumed to be already present in the communication buffers of the MESH driver so as to make it unnecessary to copy any data to/from application data structures.

Similar in spirit to GAMMA, PM [15] is a *kernel-level* messaging system providing a quite high abstraction level. The PM device driver moves data between application data structures and communication buffers allocated in kernel-level. PM guarantees message delivery by using "stop-and-go" flow control paired with a "go-back-N" retransmission scheme. An implementation of PM for the Essential EC 440SF Gigabit Ethernet adapter has been documented [14] with quite a low latency for a four-byte message (24.2 μs); a not very brilliant *transmission* throughput of 56.7 MByte/s has been reported though.

References

[1] M-VIA Home Page, `http://www. nersc.gov /research /FTG /via/`, 1998.

[2] Virtual Interface Architecture Home Page, `http://www. viarch.org/`, 1998.

[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 *Proc. 1999 International Conference on Parallel and Distributed Processing, Techniques and Applications (PDPTA'99)*, Las Vegas, Nevada, June 1999.

[4] P. Buonadonna, A. Geweke, and D. Culler. An Implementation and Analysis of the Virtual Interface Architecture. In *SC98*, Orlando, Florida, November 1998.

[5] G. Chiola and G. Ciaccio. GAMMA home page, `http://www. disi.unige.it /project /gamma/`.

[6] G. Chiola and G. Ciaccio. GAMMA: a low cost Network of Workstations based on Active Messages. In *Proc. Euromicro PDP'97*, London, UK, January 1997. IEEE Computer Society.

[7] G. Chiola and G. Ciaccio. Porting MPICH ADI on GAMMA with Flow Control. In *Proc. IEEE - ACM 1999 Midwest Workshop on Parallel Processing (MWPP 1999)*, Kent, OH, August 1999.

[8] G. Chiola and G. Ciaccio. Efficient Parallel Processing on Low-cost Clusters with GAMMA Active Ports. *Parallel Computing*, (26):333–354, 2000.

[9] G. Ciaccio. MPI/GAMMA home page, `http://www. disi.unige.it /project /gamma/mpigamma/`.

[10] A. Gallatin, J. Chase, and K. Yocum. Trapeze/IP: TCP/IP at Near-Gigabit Speeds. In *1999 USENIX Technical Conference (Freenix Track)*, June 1999.

[11] M. Lauria, S. Pakin, and A. Chien. Efficient Layering for High Speed Communication: the MPI over Fast Messages (FM) Experience. *Cluster Computing*, (2):107–116, 1999.

[12] inc. Myricom. GM performance, `http://www. myri.com /myrinet/performance/`, 2000.

[13] L. Prylli and B. Tourancheau. BIP: a new protocol designed for high performance networking on Myrinet. In *Proc. Workshop PC-NOW, IPPS/SPDP'98*, number 1388 in Lecture Notes in Computer Science, pages 472–485, Orlando, Florida, April 1998. Springer.

[14] S. Sumimoto, H. Tezuka, A. Hori, H. Harada, T. Takahashi, and Y. Ishikawa. The Design and Evaluation of High Performance Communication using a Gigabit Ethernet. In *ICS-99*, Rhodes, Greece, June 1999.

[15] H. Tezuka, A. Hori, Y. Ishikawa, and M. Sato. PM: An Operating System Coordinated High Performance Communication Library. In *Proc. of High Performance Computing and Networking (HPCN'97)*, number 1225 in Lecture Notes in Computer Science, pages 708–717. Springer-Verlag, April 1997.

[16] H. Tezuka, F. O'Carrol, A. Hori, and Y. Ishikawa. Pin-down Cache: a Virtual Memory Management Technique for Zero-copy Communication. In *IPPS/SPDP'98*, pages 308–314. IEEE, April 1998.