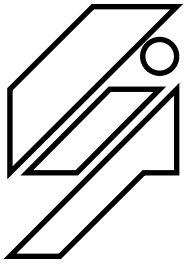
# Protocol design for high performance networking: a Myrinet experience

Loïc Prylli, Bernard Tourancheau




*Laboratoire de l'Informatique du Parallélisme*

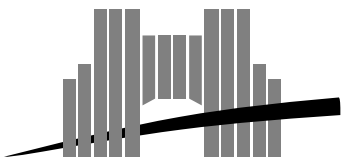
Ecole Normale Supérieure de Lyon
Unité de recherche associée au CNRS n°1398


# *Protocol design for high performance networking: a Myrinet experience*


Loic Prylli
Bernard Tourancheau


July 1997


Research Report N$^{o}$ 97-22


**Ecole Normale Supérieure de Lyon**
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : (+33) (0)4.72.72.80.00 Télécopieur : (+33) (0)4.72.72.80.80
Adresse électronique : lip@lip.ens−lyon.fr

# Protocol design for high performance networking: a Myrinet experience

Loic Prylli
Bernard Tourancheau

July 1997

**Abstract**

High speed networks are now providing incredible performance. Software evolution is slow and the old protocol stacks are no longer adequate for these kind of communication speeds. When bandwidth increases, the latency should decrease as much in order to keep the system balance. With the current network technology, the main bottleneck is most often the software that is the interface between the hardware and the user. We designed and implemented new transmission protocols, targeted to parallel computing, that squeeze the most out of a high speed network (Myrinet in this paper) without wasting time in system calls or memory copies, giving all the speed to the applications. This design is presented here as well as experimental results. We achieve Gigabit/s bandwidth and less than 5$\mu$s latency on a cluster of PC workstations with inexpensive network hardware. Moreover, our results compare favorably with the expensive parallel computers or ATM LANs.

**Keywords:** High-speed networks, Communication Protocols, Message-passing implementations, Workstations clusters, Myrinet

**Résumé**

Les réseaux haut-débits proposent maintenant des performances incroyables. L'évolution des logiciels est plus lente et les vieilles piles de protocoles ne sont plus adaptées. Lorsque la bande passante augmente, les latences doivent aussi diminuer pour maintenir un système équilibré. Avec les technologies actuelles de réseaux haut-débits, le goulot d'étranglement le plus courant est l'interface logicielle entre le matériel et l'utilisateur. Nous avons conçu et implémenté de nouveaux protocoles spécifiques, destinés au calcul parallèle, qui permettent d'obtenir des performances proches de celles du matériel (Myrinet dans notre étude). Ceci sans perdre de temps en appels systèmes ou recopies mémoires, laissant toutes les performances disponibles pour l'application. Nous présentons ici la conception et les résultats expérimentaux de l'implémentation réalisée. Nous atteignons un débit de 1Gbit/s avec une latence inférieure à 5$\mu$s sur une ferme de PC avec un réseau local d'un prix raisonnable. De plus, ces résultats de communication sont meilleurs que ceux des ordinateurs parallèles ou des stations reliées par un réseau ATM.

**Mots-clés:** Réseaux haut-débits, Protocole de communication, implémentation de bibliothèques "message-passing", Réseaux de stations, Myrinet

# Protocol design for high performance networking: a Myrinet experience

Loic Prylli
LHPC & INRIA ReMaP
LIP, ENS-Lyon
699364 Lyon - France
Loic.Prylli@ens-lyon.fr

Bernard Tourancheau*
LHPC & INRIA ReMaP
LIGIM bat710, UCB-Lyon
69622 Villeurbanne - France
Bernard.Tourancheau@inria.fr

## 1 Introduction

Multimedia applications as well as parallel computing and databases are asking for low-latency, high-bandwidth networks. This kind of performance implies a new design of the protocols in order to avoid software latency and memory copies.

Indeed, the recent relative evolution of computer subsystems has created new problems: five years ago, with parallel computing over 10Mbits/s Ethernet or even on a 100Mbits/s FDDI ring, it was easy to saturate the network. The memory bandwidth or the IO bandwidth of typical workstations were an order of magnitude faster than the physical network, so the interface between the user and the hardware was not too much a problem. Nowadays relatively inexpensive network technology is over 1Gbits/s for LAN, and although workstations have also increased in performance, the gap between network bandwith and other inner bandwiths (memory and IO busses) has been considerably reduced. So it is time to use carefully host resources.

Our experiences with ATM LAN networks have shown two problems, first even when the wire is able to provide 155Mbits/s, a poor design of the ATM board drivers can prevent the use of more than half the hardware bandwidth. Second the latency on typical workstations is counted by hundreds of microsecond [Pry96, PT97] which is unbearable in such a context (a 500 us latency is equivalent to the transfer of 10MBytes of data at the speed of 155Mbit/s).

Our software research work was driven by the idea that we wanted to exploit to its full strength the potential of the network in the applications. In the real world, what counts is not what the hardware can theoretically support (ATM 155Mbits/s, Myrinet 1.28Gbits/s) but what performance is available at the user/developer level (and which marketing will not advertise). Our research shows that the power of high-speed networks can be exploited by carefully shortening the path of data from application to application and avoiding overhead. This was necessary for both latency and bandwidth improvement.

This paper describes our ideas for the design of a software protocol that leaded to a sustained 1 Gbits/s throughput with less than 5 $\mu$s latency over a Myricom LAN of PCs.

## 2 Overview of BIP

Our first objective was to implement an interface for network communication targeted towards message-passing parallel computing. The idea was to provide protocols with low level functionalities that could be easily interfaced with classical protocol stacks like IP or APIs like the well established MPI[SOHL+95] and PVM[GBD+94] (see Figure 1).

*This work was supported by EUREKA contract EUROTOPS, LHPC (Matra MSI, CNRS, ENS-Lyon, INRIA, Région Rhône-Alpes), INRIA Rhône-Alpes project REMAP, CNRS PICS program, CEE KIT contract

The Myricom LAN target was chosen for its performance over the Gbit/s (OC-24 = 1.28 Gbits/s actually), its affordability (the interface board are around 1K USD and the $8 \times 8$ switch is around 1.5K USD) and its software openness (all the software and specifications are freely available for customers).

Our simple interface was called BIP for Basic Interface for Parallelism. Our basic idea was to build a library accessible from applications that will implement a high-speed protocol with the least possible accesses to the system kernel (for other works in this area, see Section5).

BIP provides several functions to get parameters or set-up constants, and the send and receive communication primitives. Long messages have a rendez-vous semantics : the receive must be posted before the send has begun. This is a like the MPI "ready-mode" semantic. Short messages are stored in intermediate buffers so a send call will not block even when no corresponding "receive" is posted, except when the destination buffers are full. This is a bit like the "buffered send" of MPI, the difference is that the buffer limit is fixed by the receiver, not by the sender. These primitives allow higher layers to implement any more complex semantics calls, for instance by using small control messages, we ported MPI this way.

BIP messages can be routed through multiple switches, which provides, from the OSI point of view, services that are part of the level 3 functionality (in fact Myrinet design implies that routing is done at the physical layer so we do not have to worry about it, for each message we still must provide the path that it should follow through the switches). This path is currently fixed at initialization for each pair of nodes.

Each BIP message has a tag identifying a particular receive queue on the destination, the other send arguments are the data and the logical number of the destination. The receive does not specify a particular source. Its arguments are a tag specifying a receive queue, a buffer where to receive a message, and the maximal length it can receive in this buffer.

The rationale here is that BIP is intended as a intermediate layer for other higher level protocols as soon as a complex functionality is needed. But the services provided are strongly oriented to the parallel application domain.

The send and the receive are also available with a non-blocking semantics, where one can either test or wait for the completion of the asynchronous send and receive calls. At one time, a process may only have one receive and one send posted, and not more. Send and receive operations are completely independent so you can intermix them in any manner. The non-blocking primitives allow overlapping of computation and communication when appropriate but there must be no more than one send and one receive operation pending; upper layers can provide a queuing mechanism to the user if necessary.

Here is a summary of the characteristics of BIP messages, a more complete presentations is done in the BIP manual[Pry97].

- The BIP scope of an application consists of $n$ processes numbered logically from 0 to $n-1$ at start time.
- BIP relies on **send** and **receive** primitives that have a ready-mode semantic for long messages.
- All BIP communications are reliable and ensure in-order delivery.
- For simplicity, the transmitted data must be contiguous and aligned on 4-byte word boundaries.
- BIP does not put any limit on the size of messages transmitted.
- There is several "queues" on a each destination, each message is sent to a particular queue identified by a tag. Messages on different queues can be retrieved in a order different from the physical reception, but messages are retrieved in-order on a given queue .

# 3 Design choices and methodology

The software design was in particular guided by the high-speed network platform we use (Myrinet from Myricom Inc), but the general ideas are applicable to any network hardware architecture that provide the same assumptions. We tried to present it this way and give technical hints only when it is necessary to illustrate the ideas.

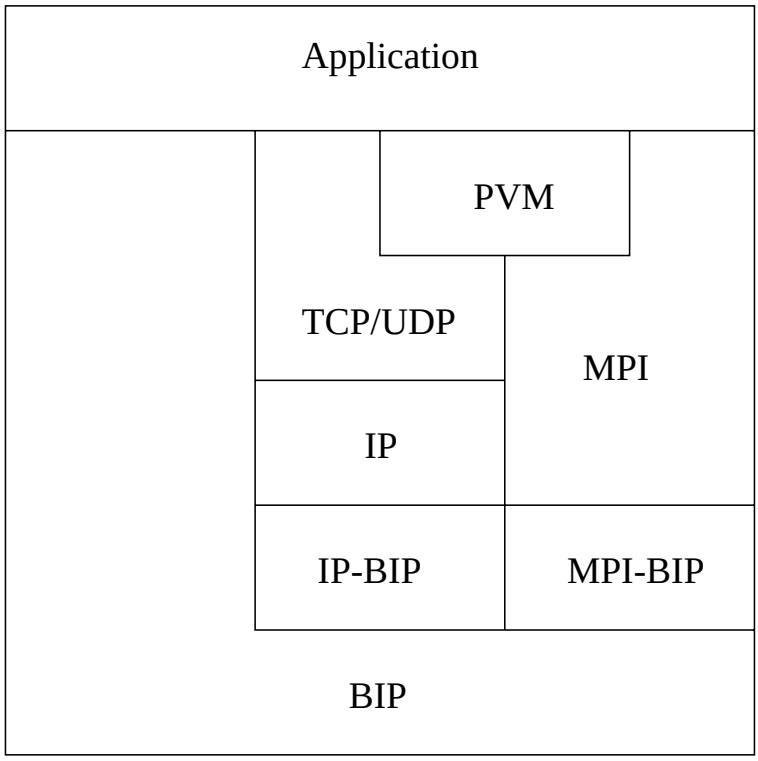



Figure 1: Description of the protocol stack with our application performance point of view: the application can access directly the BIP level (basic message passing interface with constraints) or use other functionality levels. Notice that this can be changed depending on the implementations that are done; for instance, one can imagine porting the BLACS, or TCP, or PVM directly on top of BIP in order to keep very low latency.

## 3.1 User level management of the network

The BIP software is designed for a standard UNIX configuration. We wanted to avoid implementing part of our BIP library in the UNIX kernel, but working at the equivalent of OSI level 1-3 we needed direct access to the hardware and modifications to the OS and thus use LINUX on each host of the LAN.

Let us justify user-level management. The primary aim of an OS is to isolate applications, one from another, to give them an interface to interact with the external world (I/O, IP), and to share the resources of a machine (memory and CPU) between the different applications. Security checks and context switches make systems calls a heavy operation, and for performance reasons all interaction that is contained within an application should not interact with the OS. We can illustrate this with the standard `<stdio>` that allow buffering inside a process for performance reasons, or the effort that have been made to implement user-level threads using the system only when strictly necessary.

A parallel application, although consisting of several processes on different computers, is just *one* application. As such when doing just computation it should not need to interact with the OS. Moreover, message-passing in a parallel application should not need the OS like memory copy does not need the OS in a sequential application. For this purpose, we consider the network hardware totally dedicated to the application [1].

The important point here has already been pointed out by others [PKC97, BBVvE95]: for performance reasons, we cannot afford using the OS and a heavy protocol stack as an intermediary to access the network hardware. The message-passing library should directly manage the hardware to implement the message-passing API.

## 3.2 Zero memory copy

Five years ago, on the parallel machines, or on cluster of workstations with 10Mbits/s Ethernet, the memory bus was one or several orders of magnitudes faster than the network. There was not too much concern on the way to put data from the memory to the network. Recopy of communication buffers in a memory space suited for the communication protocol, packet disassembly and reassembly, all these kind of operations did not have an important impact on the final performance.

The current situation is quite different. The network bandwidth is often comparable with the memory bandwidth (for instance, with our configuration the memory bus throughput is 180MBytes/s when reading memory, a memory copy is about 80Mbytes/s, and the network bandwidth is 160Mbytes/s). So the alternative between doing a memory copy versus putting directly user data directly on the network can impact performance as much as the old-fashioned store and forward strategy slow down routing compared to circuit-switched or wormhole strategies. In fact doing a memory copies is exactly like having a store-and-forward step on the network.

### 3.2.1 Hardware requirements

Avoiding memory copy put some requirements on the type of hardware. Not all machines are in fact capable of dealing with this situation efficiently.

For most high-speed network hardware, at some point there is a DMA from (resp. to) memory to (resp. from) either directly on the physical network or to a special memory on the interface boards (Myrinet is in the latter case).

Transfering data directly from user data space means first that the DMA engine will be able to address without overhead any location in main memory (which is not always the case, or requires an initialization with non-negligible cost).

Second, all processors nowadays have memory caches, and it will occur frequently that the valid user data is in a cache and has not yet been updated in main memory. The system bus of the machine should ensure that coherency is respected between DMA memory access and the processor cache[BS95]. That means the processor should do snooping on the system bus and provide data from its cache when it is more recent,

[1] Note that like in the U-Net project the processor on the board could multiplex transmissions between several applications. This is out of the scope of this paper but might be future work

instead of letting the memory provide it[2]. So for example, this requirement is fullfiled on most machines based on Pentium, Pentium Pro, SuperSparc, UltraSparc, ..., but MicroSPARC have not this capability.

#### 3.2.2 Software and MMU issues

The second problem for a direct transfer of user data from/to the network is the way the hardware and the OS deals with the duality between physical/virtual addresses. Basic OS system considerations make the use of paging unavoidable for common applications, so it not possible to suppose equality between physical and virtual addresses. Moreover a "contiguous" chunk of data in the virtual address space will not be contiguous in physical memory. Hence, the user data to be sent can be scattered in physical memory and it will be necessary to take care of data in blocks of one page when they can be swapped out by the OS.

Our experimental testbed consisted of PCs for which the PCI bus can directly see the physical memory without translation[3]. So before a transfer of user data, the user area is converted into a list of pairs (virtual address, block length) and locked in memory to ensure some pages cannot be swapped out.

This is done by a kernel module that is part of the BIP implementation and thanks to the LINUX design does not need a kernel recompilation. This module first provides to the user some system calls to pin pages in physical memory (mark them unswappable)[4]. Second, it allows the conversion of a virtual address into a physical memory address (after checking its page has been marked unswappable; if not the physical address could be changed or become invalid). So these functions provide a means to give the appropriate addresses to the network board DMA while transfering user data.

We said previously that we wanted to avoid costly system calls, but we needed these calls to translate virtual addresses and mark pages unswappable. A heavy solution would have been to mark the process virtual address space unswappable once and for all and to recopy the page table that contains the virtual/physical translation into user space. The solution we adopted is to do that only on demand, which means the first time a user process needs to transfer a page it is marked unswappable and its physical address is kept in memory in order to be reused latter without a system call. This avoids locking more pages than necessary and only the first messages of the application have the system call overhead.

#### 3.2.3 Network reliability issues

#### 3.2.4 No sharing between applications

Like for most parallel machines, the BIP implementation monopolizes the network interface of one processor to insure full performances. In particular there cannot be several independent BIP applications using the same processor, or the network interface cannot be used for IP traffic while the BIP application is running.

BIP is also not intended to be fair versus other UNIX processes. It is optimized for the case where the machines are temporarily dedicated to one application and it makes heavy use of busy loops and memory pinning that would severely affect other applications performance. However the system does not need any special configuration, and besides the presence of a CPU hungry process, it still runs like a normal UNIX system.

### 3.3 Dealing with big messages

#### 3.3.1 Data path limitation

When a message is sent between two computers, the data should be transmitted from main memory to the network board (step 1), then should be put on the wire (step 2). It will be received on the destination in the board (step 3) and then should be transferred to the final destination in main memory (step 4). With the previous technology, the transfer steps between the main memory and the network board were neglected,

[2] This mechanism is already necessary when you have several processors sharing the main memory, so in general all processors/architecture that have multiprocessing capabilities deal transparently with this problem.

[3] On some architecture (Sparc), the DMA can be programmed with virtual addresses, but there is in fact an IOMMU that must be programmed before a transfer with the physical address of the user data.

[4] It allows an unprivileged user to use the "almost standard" **mlock** system call with some security checks

nowadays on some platforms this is the main bottleneck, or it is of the same order as the speed of the network.

For instance, on our testbed, the hardware allows transfers on the PCI bus at about 130MBytes/s and the main memory bandwidth is a bit greater than that so it can sustain this rate. The current Myrinet/PCI board maximum throughput on the wire is less than 132MBytes/s. Hence, although the data on the wire could be cadenced at 160Mbytes/s, there are two steps of transmission that are done at less than 132MBytes/s. Clearly the peak theoretical throughput of the platform is 132MBytes/s. Moreover achieving something close to this rate assume that all the transfer operations (step 1: source host RAM to network emission board, step 2: emission board to wire, step 3: wire to destination board, step 4: destination board to destination host RAM) are fully pipelined. In our case, that means at the same time four simultaneous DMAs, two on the sending host and two on the receiving host !

Notice that this is not taking into account the presence of intermediate switches. From our knowledge, the delay introduced can be neglected. For instance the Myricom switch is a wormhole switch, so in absence of contention, the only visible effect is a very low latency (less than 100$\eta$s overhead).

### 3.3.2 Pipeline transmission

In order to approach as much as possible the fully pipelined case, we decompose the message into packets of equal size, depending of the total length. The host processors are only involved at the transfer initialization to give to the board the location where to take and respectively store the target messages. After that all the transmission is managed by our protocol implementation on the boards.

In BIP, each packet is transmitted in four steps as described above, if there is $n > 4$ packets, the transmission will be fully pipelined from the start of the fourth packet until the start of the last packet. In a first approximation one can consider each step is done at the same speed (corresponding to the lowest performance link in the transmission chain). A complete model of the transfer that matches with the experimental delays was constructed in order to compute the optimal size of the packets.

Achieving a message transfer at high speed strongly depends on its length because of the pipelines that would be introduced by the operation. The BIP protocols adapt as a function of the message length and try to maximize data pipelining and use memory copy when it is faster than DMA. This is shown on Figure 4.

Note that a message contiguous in user memory will not be contiguous in physical memory. Moreover, the alignment has no reason to be the same on the sending and receiving side, which means the splitting into packets at both sides is not the same, a packet at the receiving side may be composed of two fractions of consecutive packets. Our algorithm arranges for the flow of bytes on the wire be continuous. Alignment has a negligible impact on performance as soon as the number of packet is big enough. This is generally the case because the packet size depends on the whole message length.

## 3.4 Dealing with small messages

When dealing with very small messages, the initialization time to exchange the information between the board and the host becomes predominant over the transmission time itself. It should also be clear that the message will be transmitted in only one chunk.

At the BIP level, the cost of determining the DMA parameters by conversion of the virtual address to the physical address is bigger for small messages than the classical memory copy. Hence, for small messages there is a point where our algorithm starts to do memory copies to a fixed communication buffer to or from the board in order to decrease latency.

## 3.5 Security

The BIP implementation does not protect the system against a malicious user. In particular the user can reload the NIC software (commonly named MCP on Myrinet).

The implementation works almost totally at the user level and so the network board memory and registers are mapped into the user space. A consequence of giving the user access to the board, is that all system securities can be broken (at least by a malicious user), giving him an easy way to become root (note that this is also the case with other fast protocol implementations [PKC97]). But in fact nothing in BIP design

prevents the implementation of a secure version of the BIP protocol. This will be more code, but we think that it could be done without losing too much performance because it only require the OS protecting some data structures either on the LANAI SRAM or on main memory from the user with the MMU.

# 4 Communications benchmarks

The validation of our design is in the performance it delivers on the PCs connected to a Myricom LAN. In this Section, we present the results in two parts, latency in the protocol processing and throughput achieved with the BIP messages.

## 4.1 Latency

We measured the latency of our implementation by sending 1000 messages for small payload sizes, sending them back and measuring for each the round trip delay and dividing by $2^5$. The result we choose is the median, i.e. the 1000 values are sorted and the result is the 500th value. This seems very fair to us because it gives what the user can expected, taking apart extreme values. Typically the difference between the minimum and the maximum timing were within a few percent after some warmup.

The results obtained represents less than 1000 cycles of the 200 MHz processors: with such small timings each instruction counts !

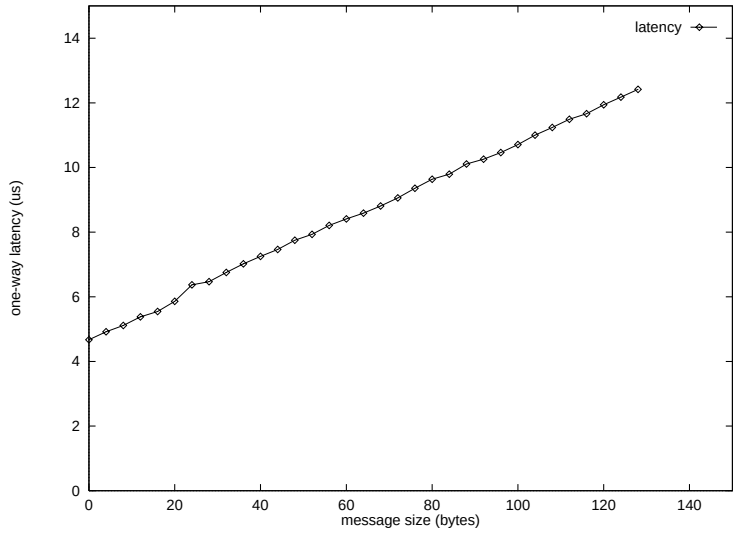



Figure 2: Latency of BIP messages for small messages on Myrinet - PCs platform.

## 4.2 Throughput

We measured the throughput of our implementation as a function of the message size. Each timing is obtained by sending 100 messages of the same size, sending them back and measuring for each iteration the round trip delay divided by $2^6$. The message size is divided by this timing to obtain the bandwidth, again we keep the median result. The Figure 3 shows the very good behavior of the protocol that reach half of its speed for small messages (4K bytes). The maximum throughput is 126 MBytes/s which represent more than 96% of the 132MBytes/s theoretical speed of our testbed.

[5] Timings are done with the Pentium cycle counter that has is incremented at each cycle count (every 5ns at 200Mhz), and each measure has a 200ns overhead. This allows to time individual round-trip.

[6] In fact we use exactly the same program than for the latency measurements

Figure 4 shows the impact of the adaptative strategy that maximize the pipeline effect. Thus the performance increases very rapidly. Figure 3 gives the maximum throughput asymptote.

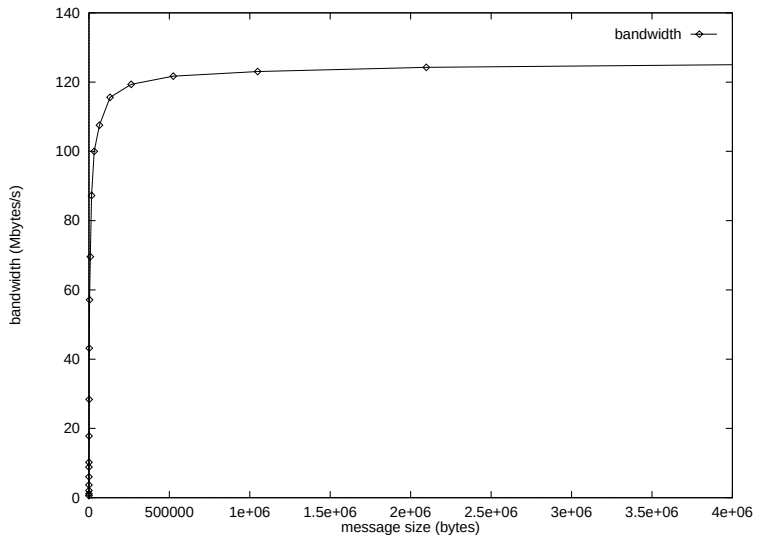



Figure 3: Throughput of BIP messages on a Myrinet - PCs platform.

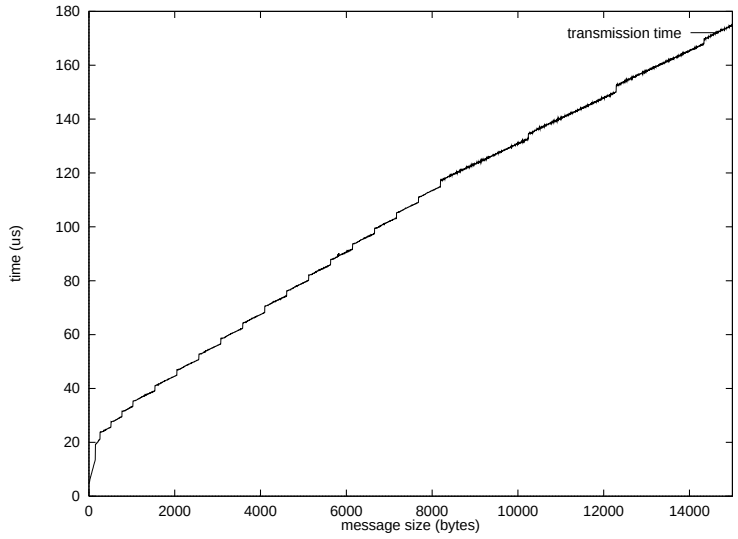



Figure 4: Transmission time of BIP messages, this curve illustrate the adaptive packet size strategy.

The table below compares our results with the benchmark of [DD95] on different parallel machines.

# 5 Related work

Our work is directly in the ideas that motivated "Active Messages" or "Fast Messages" to get rid of the costly protocol stack and system calls.

Our answer is different from the others since from the beginning we tried to focus on the idea of performance for the application, without any compromise. Another major difference is that we do not intend to

| machine | latency ($\mu s$) | Bandwidth (MBytes/s) | size of $\frac{1}{2}$ bandwidth |
|---|---|---|---|
| ATM155/sparc5 (AAL5)[PT97, Pry96] | 500 | 10 | 10000 |
| iPSC/860 (NX))[DD95] | 65 | 3 | 340 |
| TMC CM-5 (CMMD)[DD95] | 95 | 9 | 962 |
| Intel Paragon (NX)[DD95] | 29 | 154 | 7236 |
| Intel Paragon (MPI)[GLDS96] | 40 | 70 | $\tilde{7}000$ |
| Meiko CS2)[DD95] | 83 | 43 | 3559 |
| IBM SP-2 (MPI)[DD95] | 35 | 35 | 3262 |
| T3D (shmem)[DD95] | 3 | 128 | 363 |
| T3D (MPI)[GLDS96] | 21 | 100 | $\tilde{7}000$ |
| SGI Power Challenge (MPI)[GLDS96]) | 47 | 55 | $\tilde{5}000$ |
| Myrinet/Ppro200 (BIP) | 4.3 | 126 | 4096 |

Table 1: Comparison of communication performance for some parallel architectures.

provide any other paradigm of message-passing. BIP has been designed specifically as a base to implement efficiently MPI and PVM. Although other work has shown that they can be used with success for this aim, such a task is easier to do with BIP. A first version of MPI on top of BIP has been realized and run all the NAS benchmarks.

Active Messages from Berkeley[vECGS92] provide the RISC-type communications with a handler activation at the receiving side. We are not providing such a handling facility. We propose an optimized data path, an adaptative protocol and low-level pipelining.

Fast Messages from Illinois[PKC97] include a complex flow-control protocol in order to manage the communications at the user level. We are just providing a high-speed low level interface, everything else is left to upper layers.

U-net from Cornell University[BBVvE95] provides the whole protocol stack in the user space to avoid system calls. We allow to the BIP interface the two memory management necessary system calls and do not put other heavy protocols in the user space, thus no security check is ensured in BIP.

UHN-net*[HRKQ96] gives a shared data space for the user and the system by rewriting the OS kernel. And producer/consumer synchronisation of message in this buffer between the kernel and the user. We do not provide such deep modifications of the kernel, just a mechanism that allow to choose which pages are shared so avoiding memory copies.

# 6 Conclusions and future work

The project home pages are available on the net
http://lhpca.univ-lyon1.fr

We are actually working on the upper layers that will be interfaced with BIP. These works introduce feedback to our BIP functionality and we are discussing which of them should be supported at the BIP level.

The MPI library is currently ported with the MPICH Channel interface. Our first results shows only a 10 % loss in performance for the basic MPI communications compared to BIP which, to our knowledge, gives us the best throughput and latency for MPI against any other machine.

The IP stack is our next goal in order to provide the TCP/IP support at higher speed and we got actually promising results at 35MBytes/s[7].

From our design and experience, we set up the following rules:

- High speed networks required new protocols in order to give performance at the application level,
- Never do something that is not optimal in term of the number of operations for memory and data transfer on the bus (use burst-transfer on PCI systems for instance),

[7] With the same measurement method used for BIP but by send/recv calls on TCP sockets instead of BIP calls

- Model all you can in order to understand it and have the best optimization.
- Uses different strategies when performance requires them.

Finally, the raw performance numbers show that the new networking technology is going to change the way of thinking, especially when one realizes that disk access now becomes more than one order of magnitude costly than access to remote MegaBytes of RAM. It opens new opportunities in the field of distributed file systems, new opportunities for process migrations, and so on. It would be now possible to design a cluster of workstations viewed as just one server and efficiently migrate processes to do load balancing. For parallel computing, it extends the field of possible applications, low latency allowing a finer granularity parallelism. General communication performance permits the scalability of applications to larger configurations.

Our future works will concern implementation of higher level APIs (MPI or PVM), secure version of BIP (in the UNIX security sense), and multiplexing of several BIP applications on one network board.

As our work is progressing with higher communication functionality, we are enlarging BIP with the services that are not decreasing the performances. Our prototype now includes tags and a queuing facility for small messages.

# References


[BBVvE95] Anindya Basu, Vineet Buch, Werner Vogels, and Thorsten von Eicken. U-Net: A user-level network interface for parallel and distributed computing. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP)*, Copper Mountain, Colorado, December 1995.

[BS95] Gilles Berger-Sabbatel. PVM and ATM networks. In Dongarra, Gengler, Tourancheau, and Vigouroux, editors, *EuroPVM*. Hermes, 1995.

[DD95] Jack Dongarra and Tom Dunigan. Message-passing performance of various computers. Technical Report CS-95-299, University of Tennessee, July 1995.

[GBD+94] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Mancheck, and Vaidy Sunderam. *PVM: Parallel Virtual Machine*. Scientific and Engineering Computation. MIT Press, 1994.

[GLDS96] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. Technical report, Argonne National Laboratory, 1996.

[HRKQ96] Philip J. Hatcher, Robert D. Russell, Santhosh Kumaran, and Michael J. Quinn. Implementing data-parallel programs on commodity clusters. In *the Spring School on Data Parallelism*, March 1996.

[PKC97] S. Pakin, V. Karamcheti, and A. Chien. Fast messages (FM): Efficient, portable communication for workstation clusters and massively-parallel processors. *IEEE Concurrency*, 1997.

[Pry96] Loïc Prylli. Calcul parallèle sur réseau ATM de stations de travail. In *Renpar'8*, 1996.

[Pry97] Loic Prylli. *BIP User Reference Manual*, April 1997. http://www-bip.univ-lyon1.fr/software/bip-manual.ps.

[PT97] Loïc Prylli and Bernard Tourancheau. Parallel computing on an ATM LAN. In *ATM97*, Bradford, UK, 1997. IFIP.

[SOHL+95] Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI: The Complete Reference*. MIT Press, 1995.

[vECGS92] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active messages: a mechanism for integrated communication and computation. In *Proceedings of the 19th Int'l Symp. on Computer Architecture*, Gold Coast, Australia, may 1992.