User–Space Communication: A Quantitative Study

Soichiro Araki[1], Angelos Bilas[2], Cezary Dubnicki[2],
Jan Edler[3], Koichi Konishi[1], and James Philbin[3]

[1] C&C Media Res. Labs., NEC Corp., Kawasaki, Japan
[2] Department of Computer Science, Princeton University, Princeton, New Jersey 08544
[3] NEC Research Institute, Inc., Princeton, New Jersey 08540

{soichiro, konishi}@ccm.cl.nec.co.jp,
{bilas, dubnicki}@cs.princeton.edu,
{edler, philbin}@research.nj.nec.com

Abstract

Powerful commodity systems and networks offer a promising direction for high performance computing because they are inexpensive and they closely track technology progress. However, high, raw–hardware performance is rarely delivered to the end user. Previous work has shown that the bottleneck in these architectures is the overheads imposed by the software communication layer. To reduce these overheads, researchers have proposed a number of **user-space** communication models. The common feature of these models is that applications have direct access to the network, bypassing the operating system in the common case and thus avoiding the cost of send/receive system calls.

In this paper we examine five *user–space* communication layers, that represent different points in the configuration space: Generic AM, BIP-0.92, FM-2.02, PM-1.2, and VMMC-2. Although these systems support different communication paradigms and employ a variety of different implementation tradeoffs, we are able to quantitatively compare them on a single testbed consisting of a cluster of high–end PCs connected by a Myrinet network.

We find that all five communication systems have very low latency for small messages, in the range of 5 to 17 μs. Not surprisingly, this range is strongly influenced by the functionality offered by each system. We are encouraged, however, to find that features such as protected and reliable communication at user level and multiprogramming can be provided at very low cost. Bandwidth, however, depends primarily on how data is transferred between host memory and the network. Most of the investigated libraries support zero-copy protocols for certain types of data transfers, but differ significantly in the bandwidth delivered to end users. The highest bandwidth, between 95 and 125 MBytes/s for long message transfers, is delivered by libraries that use DMA on both send and receive sides and avoid all data copies. Libraries that perform additional data copies or use programmed I/O to send data to the network achieve lower maximum bandwidth, in the range of 60-70 MBytes/s.

Keywords: User–space communication, Active Messages (AM), Fast Messages (FM), Basic Interface for Parallelism (BIP), Virtual Memory Mapped Communication (VMMC), PM, LogP, Performance analysis, Latency, Bandwidth, Clusters of Wordstations, Myrinet, High performance computing.

1 Introduction

The recent arrival of fast networks, such as Myrinet [5], ATM [23, 24], or the DEC Memory Channel [12], coupled with powerful PCs and workstations, has allowed for cost-effective high-performance clusters to be built from commodity parts. This new architecture tracks the technology curve very well. Additionally, the new cluster platform offers enough raw hardware performance to allow execution of applications such as parallel scientific computing or parallel servers, which used to run on MPPs and SMPs. However, to achieve good results, the cluster communication subsystem must deliver user-to-user performance at least comparable to that of MPPs.

These new networks have exposed many problems with the traditional implementation of communications software; dramatic increases in hardware speed have not been translated to high application performance [13]. For example, the Myrinet network is capable of delivering about two orders of magnitude higher bandwidth than traditional Ethernet networks. The Myrinet implementation of the Berkeley socket interface [20], however, delivers only one order of magnitude higher bandwidth than the Ethernet implementations: a factor of 10 less than expected, due mainly to data copying and buffer management. Similarly, experiments at Cornell [2] have demonstrated that end-to-end latency on new networks, such as ATM, can sometimes exceed that of old Ethernet platforms, because of processing overheads introduced by the new interface.

The performance problems of implementing old communication interfaces (like sockets) on top of the new networks, coupled with the need for a high-performance communication layer for new classes of applications, has motivated the emergence of **user–space** communication. Previous work has shown that most overhead in traditional communication systems is caused by software, e.g., system calls, buffer management, and data copying. To avoid these costs, several thin user–space communication layers have been proposed, which eliminate system calls and reduce or eliminate buffer management overheads. These systems can be used to implement traditional compatibility libraries, compilers, and even applications code written directly by users. However, usability and applicability are strongly affected by subtle issues in the design and implementation of these software layers. High performance often conflicts with other important goals, such as multiprogramming, protected communication, portability, message ordering, fault tolerance, availability and security.

In this paper we investigate five new user–space communication libraries, Generic AM, BIP-0.92, FM-2.02, PM-1.2, and VMMC-2, on a cluster of Pentium Pro PCs running the Linux operating system and interconnected by a Myrinet network [1]. We use the methodology of [8] to quantify the LogP model parameters and give bandwidth measurements for each library. By providing a comparative analysis on a single platform, we gain significant insight into how performance is affected by the programming model and implementation choices of each library.

There are two other user–space systems that we do not investigate in this work, namely the Myricom API [5] and U-Net [2]. These high–performance systems have been developed with somewhat different goals than the communication layers we examine. The Myricom API is a general purpose library that provides basic functionality. U-Net aims at user–space, high performance communication in a broader spectrum of networks.

The rest of the paper is organized as follows: In Section 2, we present an overview of the five alternative user–space communication layers. In Section 3, we present the test-bed we used. Section 4 describes the testing methodology, presents the results and identifies the issues that most directly impact system performance. Several issues related to the study are discussed in Section 5. Finally, conclusions are drawn in Section 6.

2 Overview of User–Space Communication Libraries

In this work we evaluate five user-space communication libraries: Generic AM, BIP-0.92, FM-2.02, PM-1.2, and VMMC-2 that share many common features: all provide direct user access to the network, try to eliminate data copies, support the single-program, multiple-data (SPMD) computing model, and run on standard operating systems, like Linux, NetBSD or Windows NT. However, there are significant differences between these libraries as well. Table 1 compares the programming models implemented by the five libraries. In the remainder of this section we describe each library in more detail, referring to Table 1 to emphasize differences across the models. Differences relating solely to implementation choices are discussed in later sections.

Generic Active Messages (GAM) [11] provide the user with a model where control and data transfers are integrated. Each message specifies a remote handler that will be run upon receipt of the message. Restrictions are imposed on the operations that can be performed in the handlers. Each request is matched with a reply. Every network operation involves a round trip message exchange. The implementation of GAM we used does not tolerate network errors. Flow control is used to avoid dropping messages. Each sender has a number of tokens that it can use to send messages to a particular receiver. If these tokens are consumed, the sender must wait for replies before it can send more messages. Messages are delivered in FIFO order. There is no support for multiple processes per node. GAM provides protection within a node, but allows arbitrary functions to be called as handlers on the remote node; therefore, the receiver needs to trust the sender. There is a new, revised version of AM, AM-II, that deals with many of the limitations of the original design [16], but it does not yet run on our hardware platform. Table 1 includes both GAM and AM-II, although we have only measured the former.

Basic Interface for Parallelism (BIP) [19] is a minimal library that aims at providing raw hardware performance to its users. To achieve this, it allows direct access to all system resources and provides data transfer only (no transfer of control). It is intended for single-user systems, and supports neither protection nor multiprogramming. BIP delivers messages in FIFO order. We use version 0.92 of BIP.

Fast Messages (FM) [18, 17] provides FIFO message delivery, and assumes a reliable network. Each message carries a pointer to a function that consumes the data at the receiver. Since messages need to be consumed, flow control is provided between the sender and the receiver to avoid buffer overflow at the receiver. An important part of FM is the streaming interface that allows the user to compose a message from non-adjacent data pieces in the user address space. This provides the user with scatter/gather type operations. In this study we use FM-2.02. This version of FM does not provide support for multiple processes per node. There is a new, revised version of FM, FM-2.1, that deals with many of the limitations of FM-2.02. We include FM-2.1 in Table 1 for completeness.

PM [22] uses a daemon to multiplex communications for multiple processes over each network interface. PM was designed to support gang scheduling, so most of the protection issues are avoided by having one process access the network interface at each time. PM provides in-order message delivery and flow control between the sender and the receiver, but does not tolerate network errors. For this study, we used the latest version of PM, PM-1.2. This version of PM supports two kinds of usage: a traditional send/receive model, and a remote write model. Our benchmarks use sends and receives rather than remote writes, but also make explicit calls to acquire and access the PM system buffers directly, avoiding the overhead of an additional copy to/from application buffers.

Virtual Memory Mapped Communication (VMMC) [10] is a communication model providing direct data transfer between the sender's and receiver's virtual address spaces. VMMC provides protected user–space communication in a multiprogrammed environment, and was designed to minimize the software communication overhead, especially at the receiving side. VMMC tolerates network errors and guarantees in-order message delivery. Flow control, if needed, is the responsibility of higher-level software, since the sender deposits data directly in the receiver's memory. We use the

[1] Since AM and FM were not available for our platform when we started this work, the numbers we present are from in–house ports for Linux on the x86 architecture.

	Generic AM	AM-II	BIP-0.92	FM-2.02	FM-2.1	PM-1.2	VMMC-2
Commun. Model	RPC	RPC	Send/Recv	Send/Recv	Send/Recv	Send/Recv	Direct Deposit
Control/Data Transfer	Combined	Combined	Data Only	Combined	Combined	Combined	Separate
Intranode Protection	Yes	Yes	No	Yes	Yes	Yes	Yes
Internode Protection	No	Yes	No	No	Yes	Yes	Yes
Multi–programming	No	Yes	No	No	Yes	Yes	Yes
Buffer Overflows	Prevented	Prevented	Data Loss	Prevented	Prevented	Tolerated	Impossible
Net Errors	Catastrophic	Tolerated	Catastrophic	Catastrophic	Catastrophic	Catastrophic	Tolerated
Net Management	Static	Dynamic/Full	Static	Static	Static	Static	Dynamic/Demand

Table 1: The choices made in each library with respect to model issues. Intranode protection means that a process cannot hang the host it is executing on. Internode protection means that the system supports communication between mutually suspicious parties.

latest version of VMMC, VMMC-2.

3 Experimental Testbed

In our evaluation, we use a pair of Pentium Pro systems, running at 200 MHz and using the 440FX chipset. Each system is equipped with 256 KBytes of unified second level cache, 128 MBytes of EDO DRAM[2], a fast Ethernet card, and a Myrinet network interface (Figure 1). The PCI bus runs at 33MHz and is 32 bits wide, which results in 133 MBytes/s maximum bandwidth. The operating system is Linux-2.0.24.

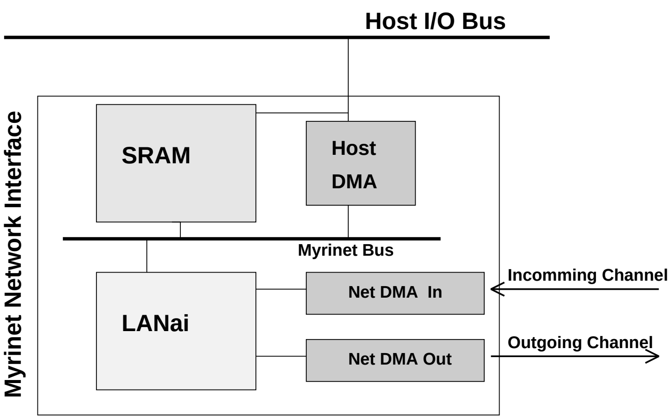

Figure 1: Myrinet network interface card.

Myrinet is a high-speed local-area network or system-area network for computer systems. A Myrinet network is composed of point-to-point links connecting hosts and/or switches. Each network link can deliver a maximum of 1.28 Gbits/s bandwidth in each direction [5].

The Myrinet network provides in-order network delivery with low bit error rates (below 10^{-15}). On sending, an 8-bit CRC is computed by hardware and is appended to the packet. On packet arrival, the CRC of each packet is computed anew by dedicated hardware and is compared with the received CRC. If they do not match, a CRC error is reported.

The Myrinet PCI-bus network interface card (Figure 1) contains a 32-bit control processor called LANai (version 4.1) with 1 MByte of SRAM [3]. The SRAM contains network buffers in addition to all code and data for the LANai processor. There are three DMA engines on the network interface: two for data transfers between the network and SRAM, and one for moving data between the SRAM and the host main memory over the PCI bus. The DMA engine that transfers data between the network interface and the host memory uses physical addresses to access host memory. The LANai processor is clocked at 33 MHz and executes a LANai control program (LCP), which supervises the operation of the DMA engines and implements a low-level communication protocol. The LANai processor cannot access the host memory directly; instead it must use the host-to-LANai DMA engine. The internal bus clock runs at twice the CPU clock speed, allowing up to two DMA engines to operate concurrently at full speed. The host can also access both the SRAM and the network itself, by using Programmed I/O (PIO) instructions. All or part of the network interface resources (memory and control registers) can be mapped to user memory.

Figure 2 shows the bandwidth of PIO and DMA for data transfers between the host system memory and the Myrinet network interface. DMA transfers between the host memory and the network interface can be initiated either by the host or the LANai processor; in both cases the DMA engine on the network interface is used. Figure 2 shows the result for both initiation methods. We see that for small and medium size messages (32 Bytes to 4 KBytes), significantly higher bandwidth is achieved if the DMA is initiated by the LANai. Moreover, in all cases, for big messages the DMA speed approaches the maximum bandwidth (133 MBytes/s) of the PCI bus and is close to this maximum even for single page transfers.

Read PIO bandwidth is lower than write PIO bandwidth because read operations on the PCI bus are more expensive than write operations. PIO performance is also strongly affected by the presence or absence of *write combining.* Write combining is a hardware feature of the Pentium Pro processor that greatly enhances write PIO performance (Figure 2) by enabling a write buffer for uncached writes, so that affected data transfers can occur at cache line size instead of word size. In the experiments reported upon in Section 4, write combining is enabled for AM and FM only.

Most user–space systems support transfers from arbitrary locations of user memory. Small message transfers can be performed with PIO, which avoids the protection and relocation problems of DMA and also results in good

[2] Extended Data Out, Dynamic Random Access Memory

[3] Static Random Access Memory

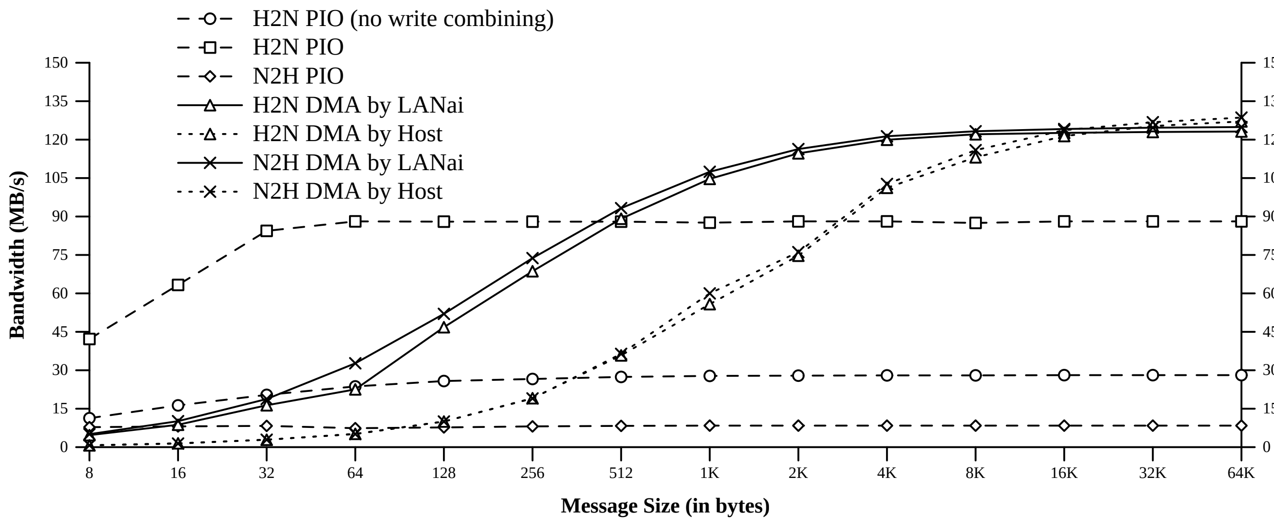

Figure 2: Bandwidth of data transfers between host memory and the network interface. *PIO* stands for programmed I/O, *H2N* stands for host memory to network interface, and *N2H* stands for network interface to host memory. DMA transfers can be initiated either by the host or the LANai processor.

performance, since it avoids the relatively high DMA initiation cost. Larger transfers (above 32–512 bytes), are more difficult to implement. One technique is to rely on write combining to reduce the cost of PIO to acceptable levels. Use of DMA incurs certain setup costs, and also requires one to deal with issues of virtual/physical address mapping and protection, since the DMA engine works with physical addresses only [21, 6]. Finally, data is transferred between the network interfaces using DMA engines on each network interface. The LANai processor has direct access to the network and it could perform the sends by copying data to control registers. This however, would tie the processor to the transfer, and it would delay the processing of other events in the network interface. For this reason DMA engines are used in all libraries to move data between the network interface and the network. In all systems that provide protection, the user is denied direct access to the DMA engine and network control registers.

4 Performance Comparison

We present numbers for the LogP parameters, $\frac{RTT}{2}, L, O_r, O_s$, and g [8], and for three different types of bandwidth, unidirectional, bidirectional ping-pong and bidirectional simultaneous. Figures 4–8 present the LogP numbers. The different types of bandwidth are shown in Figures 10–12. We perform the measurements with two nodes connected by a switch[4]. Since protection, buffer management, reliability, and data transfer techniques are critical to achieving good performance, Table 2 presents the choices made by each library with respect to these issues.

4.1 LogP parameters

The LogP parameters measure the following characteristics of a communication subsystem: the time a message spends between the source and the destination network interfaces (L), the time the host processor is involved in sending or receiving a message (O_s, O_r), and the interval between successive sends when the buffering capacity of the network is saturated (g). Since the benchmarks measuring these parameters depend on the communication model that is used in each library, in the next few paragraphs we describe how our benchmarks measure these costs and we identify some peculiarities of the measurement methodology for each library.

Our measurement methodology is derived from that of [8], and is extended to deal with the characteristics of each library under investigation. The general approach is to measure $\frac{RTT}{2}$ and g directly, then measure or calculate O_s and O_r by a method appropriate to each particular library, and finally compute L as $L = \frac{RTT}{2} - O_s - O_r$.

Flow control is an important aspect of the communications model, and its impact must be carefully considered when measuring the LogP parameters. Myrinet implements a form of STOP/GO flow control in hardware [5]. This mechanism however, will stop a sender for no more than 50 ms. After that, the sender will reset the receiver (forward reset) and communication will resume. This reset operation will empty certain queues and buffers and data may be lost.

Given this mechanism, BIP requires that, for reliable delivery of large messages (equal to or larger than 256 Bytes), the receive operation be posted before the message arrives, so that the network interface knows where the data should be transferred, and no data will be lost. If the receive is not posted in time, network resets may occur, with catastrophic results. When a message smaller than 256 Bytes arrives without a receive operation yet posted, BIP attempts to avoid data loss by making use of a fixed-size buffer in the network interface, but if the buffer is full of other messages, data loss will occur.

The other libraries avoid the problems associated with flow control by more extensive buffering or by careful buffer management in the receiver and by knowing always where messages need to be delivered. In VMMC, because mes-

[4]In our experience, the switch has a negligible effect on performance (on the order of 100ns).

	Generic AM	AM-II	BIP-0.92	FM-2.02	FM-2.1	PM-1.2	VMMC-2
Send Data	DMA+Copy	PIO(WC)	DMA+v2p trans	PIO(WC)	PIO(WC)	DMA+Copy	DMA+v2p trans
Recv Data	DMA+Copy	DMA+Copy	DMA+v2p trans	DMA+Copy	DMA+Copy	DMA+Copy	DMA+v2p trans
Protection	Copy	Copy	None	PIO+Copy	PIO+Copy	Gang Scheduling	v2p trans
Reliability	Library+Copy	Library+Copy	None	None	None	None	Firmware
Notification	Polling	Polling	None	Polling	Polling	Polling	N/A

Table 2: The choices made in each library with respect to implementation issues. The *Send Data* and *Recv Data* rows refer to the mechanism used to transfer long messages from and to arbitrary locations in user memory. All libraries, except PM, use PIO to transfer short messages. *WC* stands for write combining and *v2p trans* for virtual to physical translation [21, 6]. *Protection* refers to how each system achieves transfers to and from arbitrary locations in virtual memory, without compromising system protection. Although BIP uses virtual to physical translation, it gives direct access to control information on the LANai, and thus does not provide protection. FM uses PIO on the send side and copying on the receive side to transfer data from and to arbitrary locations in memory.

```
Barrier();          Barrier();
GetTime();          ;
rep N times         rep N times
    SendMsg();          RecvMsg();
    RecvMsg();          SendMsg();
end                 end
GetTime();          ;
Barrier();          Barrier();

      Node 0              Node 1
```

$\frac{RTT}{2}$
(a)

```
Barrier();          Barrier();
GetTime();          ;
rep N times         rep N times
    SendMsg();          RecvMsg();
    Spin(D);            ;
end                 end
GetTime();          ;
Barrier();          Barrier();

      Node 0              Node 1
```

O_s, O_r, and g
(b)

```
Barrier();          Barrier();
GetTime();          ;
rep N times         rep N times
    RecvMsg();          SendMsg();
    Spin(D);            ;
end                 end
GetTime();          ;
Barrier();          Barrier();

      Node 0              Node 1
```

O_r
(c)

Figure 3: Pseudo-code for the micro-benchmarks computing the LogP parameters. Code fragment (a) measures $\frac{RTT}{2}$ for all libraries. Code fragment (b) is used to measure O_s and g in all libraries, and O_r in AM. Code fragment (c) is used to measure O_r in BIP, FM, and PM. O_r is trivially zero in VMMC. L is calculated in all libraries from the equation $\frac{RTT}{2} = O_s + L + O_r$, after $\frac{RTT}{2}$, O_s, and O_r have been determined.

sages are directly deposited in memory without intervention by the receiver, external synchronization must be used if the messages are to be consumed without being overwritten. Our measurements use additional synchronization only for BIP, since otherwise data may be lost. The motivation for not doing the same in VMMC, even though data may be overwritten, is to avoid complicated communication schemes that may not be used by real applications. This is an example of the difficulty of making fair and meaningful performance comparisons between dissimilar communications models.

Synchronous send operations return only after the data is transferred out of the send buffer and the send buffer can be reused. Thus, the sender needs to wait for the data to be transferred out of the send buffer, which increases the cost of the send operation. Asynchronous operations return before the data is transfered out of the send buffer, and the sender can continue with useful work, without however, modifying the send buffer. The send operations we use in measuring the LogP parameters and the different types of bandwidth are synchronous for AM, BIP, and FM. The versions of AM, and FM we use do not support asynchronous operations. BIP support asynchronous send operations, but we encountered significant difficulties in using them in the micro-benchmarks. The send operations we use for PM, and VMMC are asynchronous. In general, in our benchmarks we try to use the operations with the lowest possible overhead, since these exhibit the best performance that can be seen by the user.

Another important difference among the libraries is the way in which size-dependent components in the send path are attributed to different LogP parameters. The path from the sender to the receiver is composed of the following stages: application to system buffer transfer, application or system buffer to NI transfer, NI to NI transfer, NI to application or system buffer transfer, and system to application buffer transfer. In our benchmarks, where possible, we avoid the copies and this identifies the system and the application buffers (zero copy transfers). Thus the first and the last stages in this pipeline may not be applicable in some systems. In real programs however, it is not always possible to avoid the extra copies, when sending data from and delivering data to arbitrary locations in user memory. Table 3 presents a summary of how the cost of each stage is attributed. We clarify this table where necessary in the analysis of each LogP parameter below.

Figure 3 shows the general pseudo-code of our LogP micro-benchmarks. `Spin(D)` spins for D μs. `RecvMsg()` is a blocking operation. The implementation of each function differs among the libraries. In AM, `RecvMsg()` simply spins on a flag waiting for the receive handler to receive a message, and `SendMsg()` not only sends a request, but also

	GAM	BIP-0.92	FM-2.02	PM-1.2	VMMC-2
Application to System Buffer	O_s	–	–	–	–
Application/System Buf to NI	O_s	O_s	L	L	L
NI to NI	L	O_s	L	L	L
NI to Application/System Buffer	O_r	O_s	O_r	L	L
System to Application Buffer	–	–	–	–	–

Table 3: Relationship between message transfer path stages and LogP parameters. For each component of the send path, the LogP parameter is shown, to which the cost of the component is attributed.

implicitly handles a reply, if one has arrived. In BIP, for messages longer than or equal to 256 Bytes, extra messages are sent in benchmark (b) for synchronization, so that no messages are lost. In AM, and PM, the benchmark code for sending long messages has to take care of packetization, since this is not done in the library. In VMMC, since there is no receive operation, the `RecvMsg()` call just spins on a flag in memory.

4.1.1 One way latency

$\frac{RTT}{2}$ is the one-way latency between two nodes in a ping-pong test (Figure 4). $\frac{RTT}{2}$ is measured for all libraries with code fragment (a) in Figure 3. The knees at 256 bytes for BIP and at 64 Bytes for VMMC are because of the switching from PIO to DMA. The PIO/DMA switchover effect is less visible for AM.

We see that BIP, which offers the lowest functionality among the libraries, achieves the best latency for small messages, about 6 μs for 4–Byte messages. Although a direct comparison among the different libraries cannot be very precise, we see that going to FM and PM, which adds protection and multiprogramming (in the case of PM), costs about 3 μs and increases the minimum latency to about 9μs. Adding reliability, in VMMC, costs about 2.5–3.5 μs and increases the minimum latency to about 13 μs. Finally, although AM lacks some of these features, it incurs even higher minimum latencies because of the stronger semantics of its API.

4.1.2 Gap

The gap g, is the minimum interval on the sending processor between successive sends at steady state (Figure 5), and is measured with code fragment (b) of Figure 3 when D is zero and N is large enough for the interval to reach its asymptotic limit, where for each new message sent, a message needs to be delivered at the receiver. Messages need to be consumed in AM, BIP, FM, and PM, but not in VMMC. For BIP, the sudden gap increase at 256 bytes occurs because of the externally imposed synchronization.

4.1.3 Send overhead

O_s is the host overhead of sending a message (Figure 6). For AM, O_s is directly measured with the same code fragment as g, when D is zero and N is below the network's buffering capacity. For the other libraries, where reply messages are not required, we need not limit our measurements to small N. Instead, we use the equation $g(D) = O_s + max(Idle, D)$, where as D increases from values smaller than $Idle$ to values larger than $Idle$, $g(D)$ increases as well. We measure $g(D')$ for some values D' in $\{D : g(D) > g(0)\}$, and compute O_s as the average of $g(D) - D$ for those D'.

The value of O_s depends a great deal on the method of data transfer to the network interface. The cost of PIO is included in O_s. The cost of DMA is included in O_s only with synchronous sends. By using asynchronous sends (and DMA), the processor can continue with useful work after the send operation is posted. FM always uses PIO (with write combining) to send the data, and thus sends are synchronous and depend on message size. Most of the other libraries use PIO for small messages, and DMA for large messages, but PM always uses DMA. In the case of AM, large message data is first copied to a library buffer before DMA is used; the cost of this copy is included in O_s. BIP uses DMA without extra copies, but as mentioned above, additional synchronization is imposed for messages longer than or equal to 256 Bytes. PM and VMMC use asynchronous sends (and DMA) without an extra copy, with the benefit that for large messages, their O_s results are independent of message size.

4.1.4 Receive overhead

O_r is the host overhead of receiving a message (Figure 7). O_r is computed in different ways, depending on the characteristics of each library:

For VMMC, O_r (Figure 7) is trivially zero, since the host processor at the receive side is not involved in the delivery of messages. In this case the cost of transferring the data from the network interface to user memory is attributed to L (Table 3), although memory contention between the host processor and DMA engine could still be a performance issue for real programs.

In AM, O_r is measured with code fragment (b) in Figure 3 [8]. The measurement in the sender computes $g(D) = O_s + O_r + max(Idle, D)$, since every request in AM (that incurs overhead O_s) is matched with a reply that needs to be processed (and incurs an overhead O_r). As D increases from values smaller than $Idle$ to values larger than $Idle$, g will increase as well. Thus, we can compute $Idle = D' : for\ all\ 0 <= D <= D', g(D) = g(0)$. Having computed $Idle$ and $g(0) = g$ we can then compute $O_r = g(0) - O_s - Idle$.

In BIP, FM, and PM, O_r is computed with code fragment (c) in Figure 3. The difference from code fragment (b) is that we measure the receive cost at the receiving node instead of the sending node, because there are no replies. We calculate O_r with the equation $g(D) = O_r + max(Idle, D)$, and use the same method to compute O_r as we did for O_s.

In BIP, PM, and VMMC, O_r is independent of the message size since data is transferred to the host memory without host processor intervention. For AM and FM, O_r

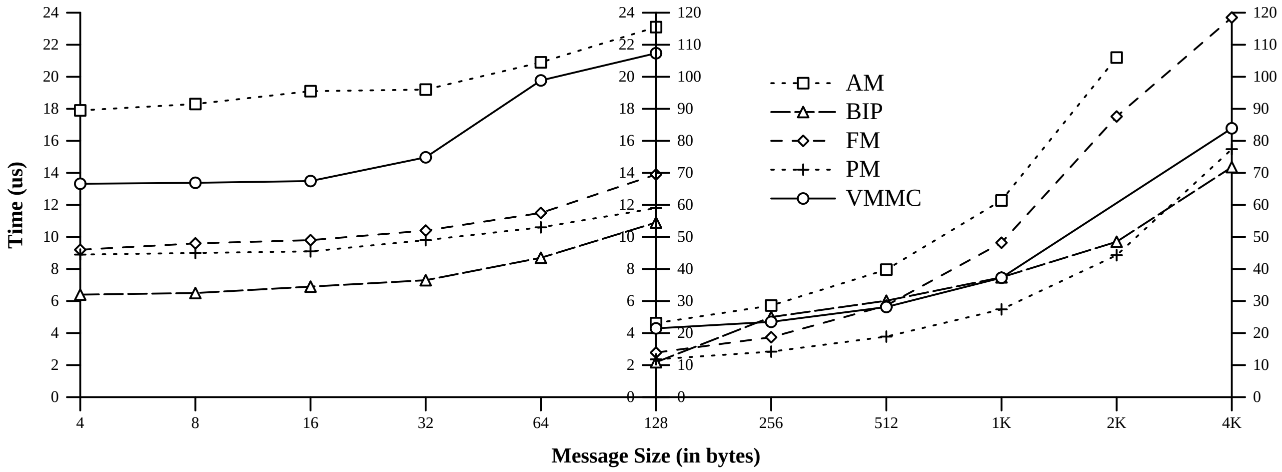

Figure 4: One way latency ($\frac{RTT}{2}$). The data point for 4 KByte messages for AM is 197.9 μs and is omitted for convenience.

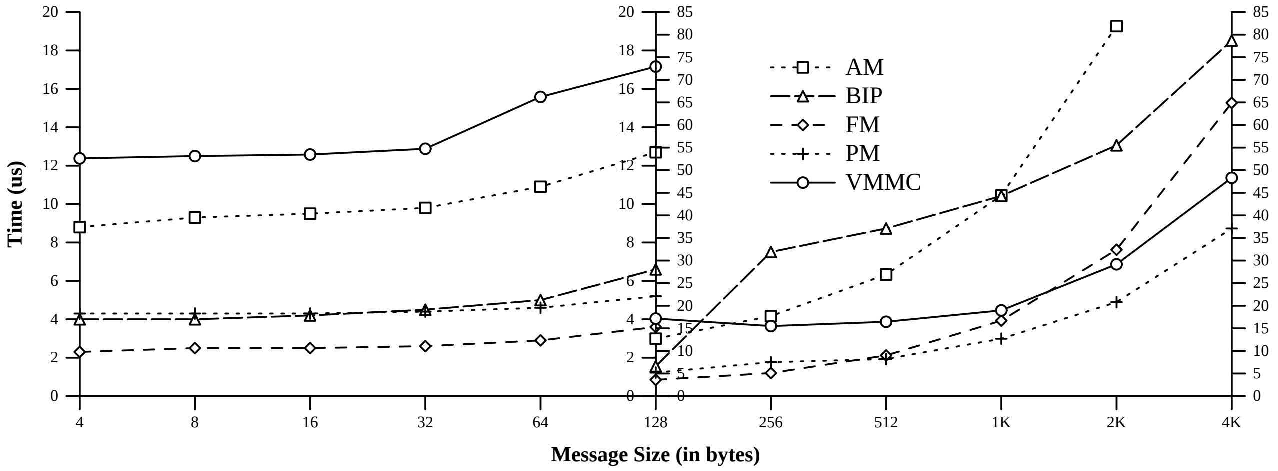

Figure 5: Gap (g) at steady state. The data point for 4 KByte messages for AM is 153.5 μs and is omitted for convenience.

depends on the message size, since the task of moving the data from the library buffer to the application buffer is part of the receive operation (Table 3).

4.1.5 Latency

In all cases, L (Figure 8) is computed from the equation $\frac{RTT}{2} = O_s + L + O_r$ [8]. Table 3 shows what costs are attributed to L. As expected, L depends strongly on message size, except for BIP, where the additional synchronization introduced attributes the cost of the NI–to–NI transfer to O_s (Table 3). The use of DMA rather than PIO also has a noticeable effect on L.

The decrease for FM at 2 KBytes is due to aggressive latency-hiding efforts in the implementation. FM carefully overlaps message sending and receiving with the transfer between the network interfaces. The receiving message handler is called when the first byte of the message arrives, while the sender may still be sending parts of the same message.

4.2 Bandwidth

We use three types of bandwidth that capture different communication patterns occurring in user programs. Pseudo–code for the bandwidth micro–benchmarks is shown in Figure 9. As for the LogP parameters, the implementation of each function differs among the libraries.

In **Bidirectional ping-pong bandwidth** (Figures 9(a), 10), data flows in both directions alternately, in a ping-pong fashion. Each node waits after sending a message until it receives one. This type of bandwidth is the reciprocal of $\frac{RTT}{2}$, scaled appropriately.

Unidirectional bandwidth (Figures 9(b), 11), is the bandwidth of a one way transfer. Data is flowing in one direction only, and the sender does not need to wait for the receiver to acknowledge a message before it sends the next one. This type of bandwidth is essentially the reciprocal of g, scaled appropriately.

In **Bidirectional simultaneous bandwidth** (Figures 9(c), 12), data flows in both directions. The senders simultaneously send and receive messages. This benchmark can take advantage of all six DMA engines of the two net-

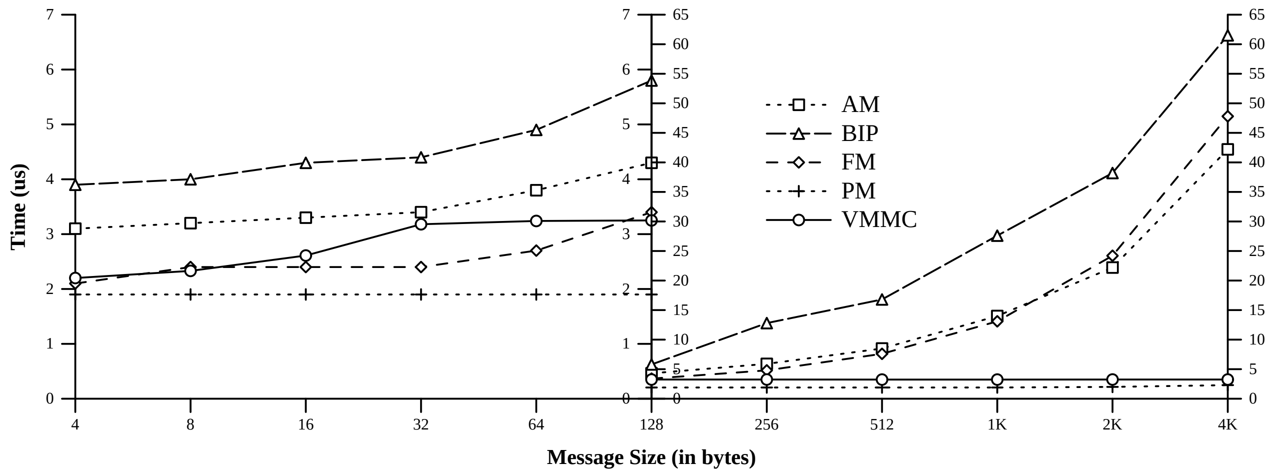

Figure 6: Send overhead (O_s)

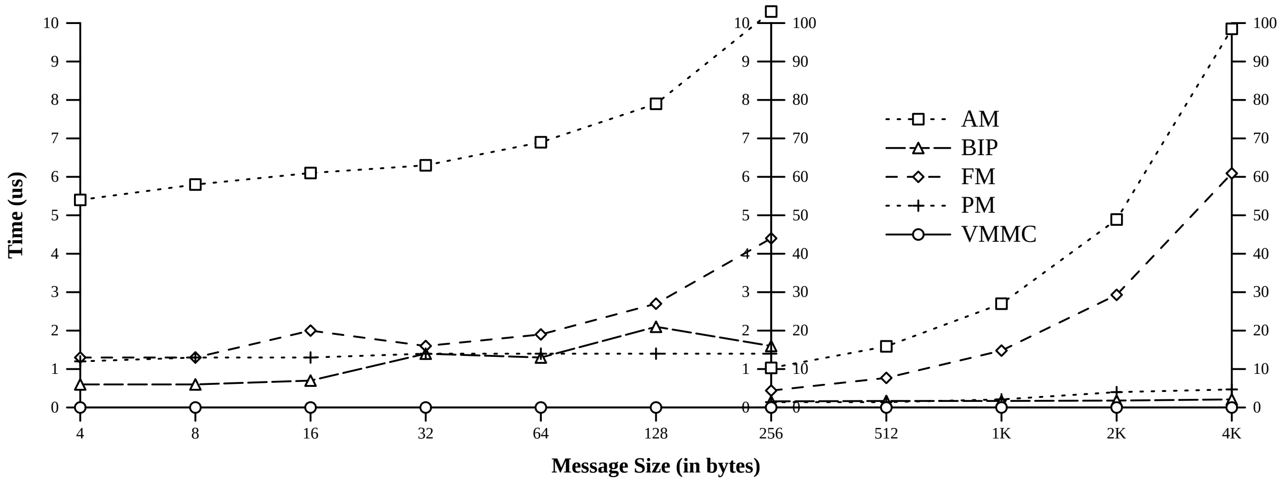

Figure 7: Receive overhead (O_r)

work interfaces, depending on how the LANai processor firmware is written.

The solid line in Figures 10–12 represents the maximum bandwidth of transferring data from the network interface to the host. This curve is composed from the curves in Figure 2, by using the *N2H PIO* curve for messages up to 16 Bytes, and the *N2H DMA by LANai* for larger messages. From the same figure, we see that the network interface to host transfers are the bottleneck, since they exhibit a lower maximum bandwidth than transfers from the host to the network interface. The solid curve in Figures 10–12 shows what the maximum bandwidth could be at steady state, and thus how well the libraries do in delivering the raw hardware performance to the end user.

In the unidirectional bandwidth benchmark, where the sender and the receiver do not explicitly synchronize, messages need to be consumed for AM, BIP, FM, and PM, but not for VMMC. In BIP, external synchronization is necessary for messages equal to or bigger than 256 bytes to avoid network resets. These extra messages are responsible for the knee at 256 Bytes in unidirectional bandwidth, by reducing the payload (more messages to transfer the same amount of data) and increasing g. The knee at 256 Bytes in BIP with bidirectional ping–pong bandwidth is due to the switching from PIO to DMA transfers, whereas the knee at 256 Bytes with unidirectional and bidirectional simultaneous bandwidths is due to both the switch from PIO to DMA and the introduction of extra synchronization messages, to ensure that the receive operation is posted before the corresponding send operation. It is not clear why there is a knee at 1 MByte messages for BIP in bidirectional simultaneous bandwidth. We also note that there is a knee at 256 KBytes, and 512 KBytes for FM and AM respectively in all types of bandwidth due to cache effects.

In general, we see that the libraries that use DMA to transfer large messages and avoid copying (BIP, PM, and VMMC) have higher bandwidth for large messages, compared to libraries that use PIO (Table 2). Although AM uses DMA to transfer the data, a copy from the user to a library buffer on the send side reduces bandwidth by 50% compared to VMMC and by about 60% compared to BIP. FM and PM have considerably better bandwidth for small and medium sized messages than the other libraries. FM benefits significantly from the use of write combining, which

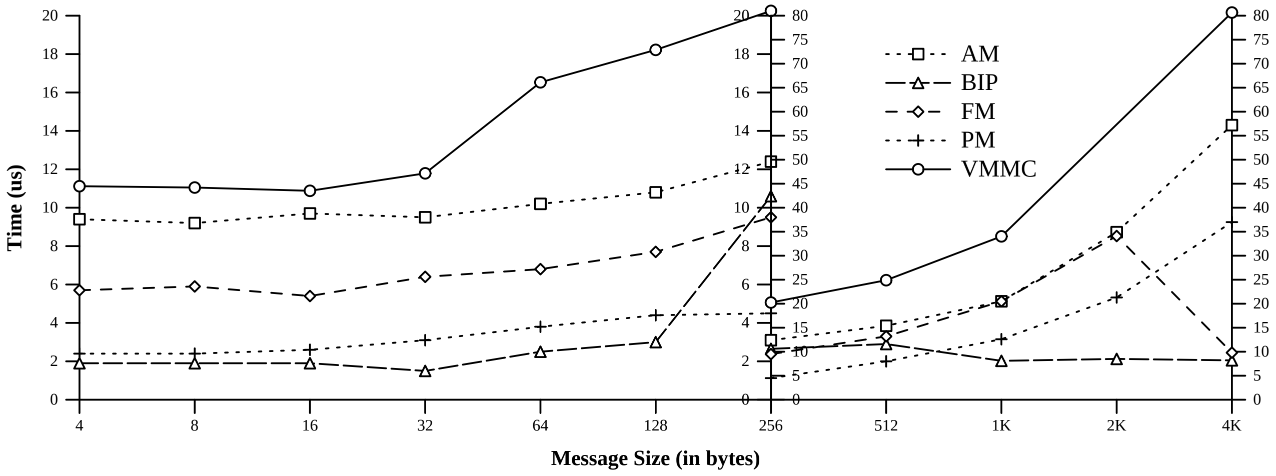

Figure 8: Latency (L)

(a) Node 0	(a) Node 1	(b) Node 0	(b) Node 1	(c) Node 0 and Node 1
Barrier();	Barrier();	Barrier();	Barrier();	Barrier();
GetTime();	;	GetTime();	;	GetTime();
rep N times	**rep** N times	**rep** N times	**rep** N times	**rep** N times
SendMsg();	RecvMsg();	SendMsg();	RecvMsg();	SendMsg();
RecvMsg();	SendMsg();	;	;	RecvMsg();
end	**end**	**end**	**end**	**end**
GetTime();	;	GetTime();	;	GetTime();
Barrier();	Barrier();	Barrier();	Barrier();	Barrier();
Node 0	Node 1	Node 0	Node 1	Node 0 and Node 1
Bidirectional Ping-pong **(a)**		Unidirectional **(b)**		Bidirectional Simultaneous **(c)**

Figure 9: Pseudo-code for the different types of bandwidth.

increases the PIO bandwidth dramatically. PM achieves high bandwidth for these message sizes mainly by taking advantage of gang scheduling and not having to support multiprogramming in the LANai code, and by using a technique they call immediate sending [22] to overlap host to NI and NI to network DMAs.

5 Other Design Issues

Table 2 presents the choices made in each library with respect to various implementation issues. The issues that are critical for performance were explained in Section 4. In this Section we discuss other important issues, that are not captured by the micro-benchmarks, but can significantly impact the usability of each system.

Data copying: In traditional communication systems, data transfers between domains have been achieved with copying. For instance, data would be transferred in turn from the network, to a system buffer, a library buffer, and, finally, the application buffer. Previous work [9] has shown that reducing the amount of copying is crucial for application performance as network performance improves. All the systems under consideration include support for zero copying protocols. This term, however, is misleading since copies can be avoided in only some cases. The performance of the libraries is strongly dependent on the extent to which copies are avoided in practice. For instance, in AM, FM, and PM, where a specific region of the memory is used as network buffer, copies can be avoided only if the application can prepare and consume the data in place. In BIP and VMMC however, copies can be avoided in more cases.

Reliable communication: As mentioned above, Myrinet does not provide the user with completely reliable hardware. Previous work has argued that reliable transmission is important for delivering high–performance to the end user [13]. There are different levels at which the communication libraries can implement reliability. Of the libraries that support reliable communication, AM-II and VMMC, AM-II choose to implement reliability in the library at the cost of an extra copy. VMMC avoids the copy by implementing reliable communication in the network interface (data link layer). This however, requires more SRAM on the memory interface and more careful buffer management.

Network management: Network routes can be computed either statically, when the system is started, or dynamically (see Table 1). Moreover dynamic mapping of the network can take different forms. It can be done either for all the

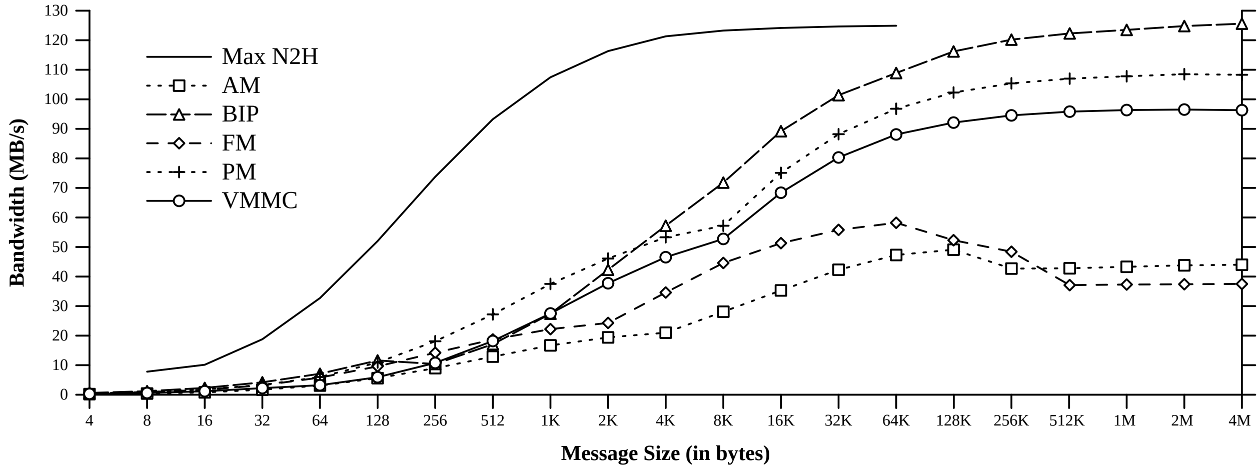

Figure 10: Bidirectional Ping-pong Bandwidth

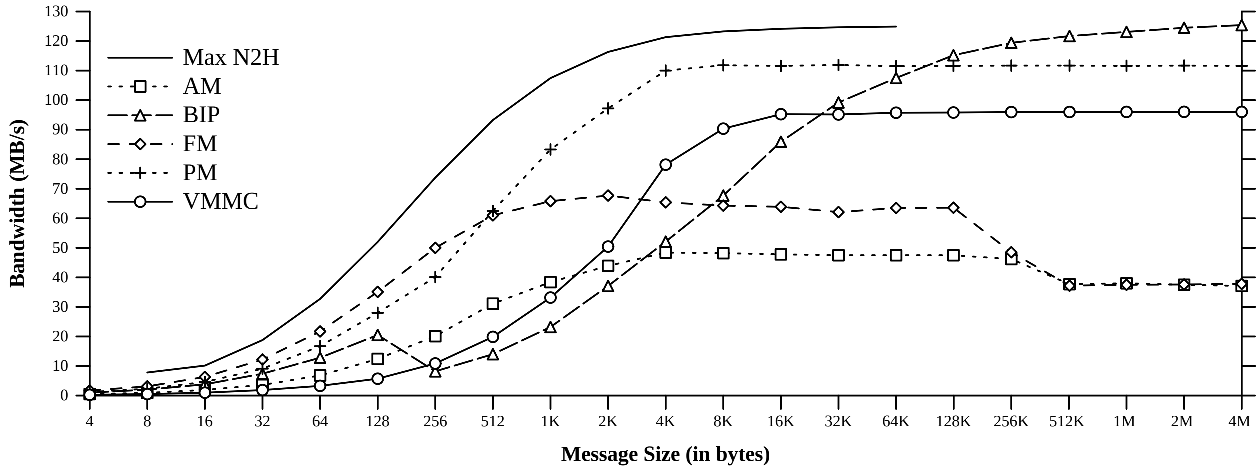

Figure 11: Unidirectional Bandwidth

nodes in the network [15] or on demand [3], when a node needs to find a new route. Dynamic network mapping and reliable communication allow for easy maintenance, upgrading, changes in the network topology, and configuration.

Connection establishment: Mechanisms for connection establishment can either be provided by a library or left to an external agent. The ability to create not only point-to-point connections but multi-point connections as well (many senders to one receiver) can impact application performance in subtle ways. For instance, in VMMC, which supports only point–to–point connections, a receiver must poll one connection for each potential sender. In a system with many senders (e.g. client–server applications), finding the next message to process may incur a high overhead. Moreover, setting up connections is usually an expensive operation, since some form of a protected channel needs to be established, either through the operating system or some other trusted entity. In cases where many connections need to be created and destroyed, this can become a bottleneck, and no library seems to solve this problem in a satisfactory way for the general case.

Interrupt Handling: The cost of delivering interrupts has been revealed to be a major problem in many cases [4, 14]. All libraries under consideration try to avoid expensive interrupts by having the network interface place messages directly in user memory. Furthermore, libraries that require handlers to run at receipt of each message, specifically AM, avoid up-calls by polling for network events at the receiver. None of the systems under consideration does a good job in reducing the cost of interrupts, mainly because the operating system can not be ignored. This favors programming models and API that minimize the number of up-calls that the user program has to handle, by providing direct access to remote memory, as opposed to a more traditional send/receive model, where action needs to be taken by the host processor at the receive side before message reception is to be completed.

LogP Model: In our study we used LogP parameters to evaluate the communication subsystems. This serves as a set of micro-benchmarks that characterize the network. The LogP model, however, was designed with other types of networks in mind. Modern networks, that take advantage

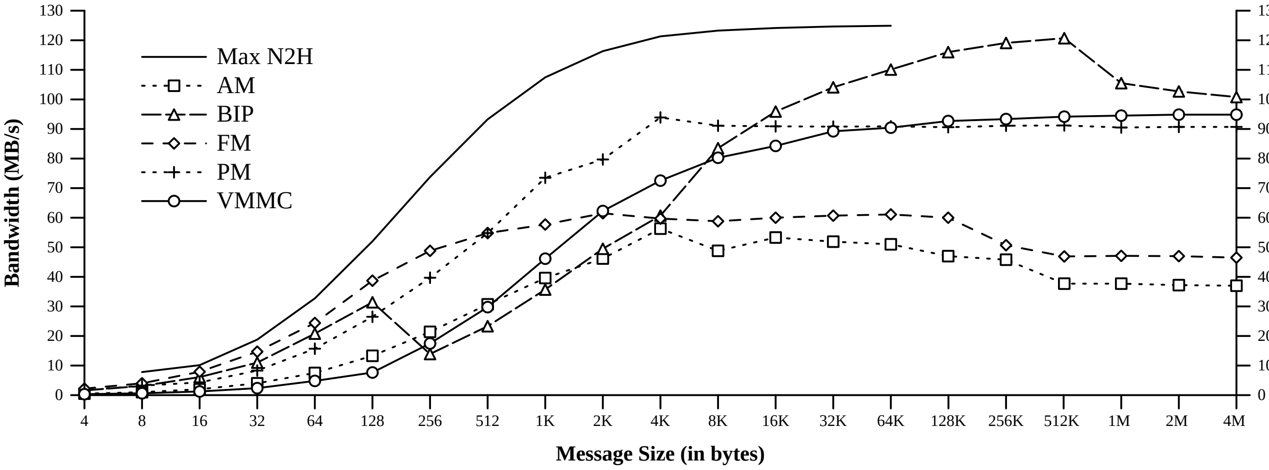

Figure 12: Simultaneous Bidirectional Bandwidth

of pipelining and more dynamic resource sharing, are more difficult to characterize in detail. The LogP parameters are not fully adequate for these networks.

In the presence of DMAs it is not clear where certain costs should be attributed — for instance a data transfer that is done with DMA is usually attributed to the network interface. However, the memory bus is occupied during the transfer, and this may interfere with the processor when it encounters cache misses.

The programming model of each system introduces additional complications. In BIP, for instance, messages may be dropped. Is g measured when messages are allowed to be dropped, or should they be received? Similarly, in VMMC, g is measured with messages being deposited in the receivers memory, but no additional synchronization is included to make sure that each message is not overwritten by the next one before it is seen by the application.

Micro-benchmarks are the only way to characterize a communication subsystem at a low level. However, experimenting with real workloads gives more insight into how applications can benefit from these systems in practice.

Other issues: Finally, there are many issues that are not addressed by any of the systems under consideration. For instance a process fork introduces a number of problems for user-space communication like dealing with copy-on-write, and connection inheritance [1]. Handling these problems may require changes to the underlying operating system, so they are not solved yet. Other issues, such as fault containment, fault tolerance, fairness, real time guarantees, quality of service [7], mapping and routing, general purpose flow control, system behavior under heavy load, etc., that have been studied in more traditional networks and parallel architectures, are either ignored or given very little consideration. Many of these issues can be handled by higher level software layers, but there has so far been little experience in actually doing so.

6 Conclusions

In this paper we explore the model and implementation space for user–space communication systems. We use the LogP model and three types of bandwidth to compare the performance of five state of the art user–space communication subsystems, Generic AM, BIP-0.92, FM-2.02, PM-1.2 and VMMC-2. Although the libraries share common goals, they have made very different decisions in both the communications model and implementation. they offer different levels of functionality and performance. The model and implementation choices lead to varying performance not only for very small or very large messages, but for the whole spectrum of message sizes and communication patterns.

We compare the performance of the five communication subsystems on the same platform and explain their behavior for each message size and communication pattern in the benchmarks. We find that short message latency lies in a range of 5-17 μs, and maximum bandwidth lies in the range of 50-125 MBytes/s. With some exceptions, an inverse relationship is observed between functionality and small message latency, with each implementation striving for the best practical compromise. On the other hand, bandwidth depends primarily on how data is transferred between host memory and the network.

Subtle issues that are sometimes ignored can impact the usability of a communication system and the performance of real applications. For instance, flow control, multipoint connections, and support for zero copy in different situations are examples of model and implementation issues that can affect application performance. The lack of reliable communication, or poor support for connection establishment can make a system difficult to use.

Finally, although the set of benchmarks that measures the LogP parameters characterizes in some sense a communication subsystem, it is not adequate. Modern network interfaces exhibit characteristics (i.e. pipelining) that are difficult to capture. Moreover, the communication model of a library (BIP, VMMC) can make it very difficult for these benchmarks to meaningfully represent the application communication patterns. We see a need for a set of higher-level benchmarks each intended to measure support for a particular application communication pattern (such as producer-consumer or central shared queue) while taking into account how the data is processed (for example, whether it is consumed in-place or needs to be copied to an arbitrary user buffer). These benchmarks would complement the LogP micro-benchmarks and offer more insight on the impact of the communication layer on application performance.

7 Acknowledgements

We would like to thank Bob Felderman for his help with the Myrinet network, Scott Pakin for his help in porting FM to Linux, and Hiroshi Tezuka and Yutaka Ishikawa for their help with PM.

References

[1] T. Anderson, B. Bershad, E. Lazowska, and H. Levy. Scheduler activations: Effective kernel support for the user-level management of parallelism. In *Proceedings of the Thirteenth Symposium on Operating Systems Principles*, pages 95–109, Oct. 1991.

[2] A. Basu, V. Buch, W. Vogels, and T. von Eicken. U-net: A user-level network interface for parallel and distributed computing. *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP), Copper Mountain, Colorado*, December 1995.

[3] A. Bilas. *Improving the Performance of Shared Virtual Memory on System Area Networks*. PhD thesis, Dept. of Computer Science, Princeton University, August 1998. Available as technical report TR-586-98.

[4] A. Bilas and J. P. Singh. The effects of communication parameters on end performance of shared virtual memory clusters. In *In Proceedings of Supercomputing 97, San Jose, CA*, November 1997.

[5] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 15(1):29–36, Feb. 1995.

[6] Y. Chen, C. Dubnicki, S. Damianakis, A. Bilas, and K. Li. Utlb: A mechanism for address translation on network interfaces. In *The 8th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 1998.

[7] K. H. Connelly and A. A. Chien. FM-QoS: Real-time communication using self-synchronizing schedules. In ACM, editor, *SC'97: High Performance Networking and Computing: Proceedings of the 1997 ACM/IEEE SC97 Conference: November 15–21, 1997, San Jose, California, USA.*, pages ??–??, New York, NY 10036, USA and 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1997. ACM Press and IEEE Computer Society Press.

[8] D. Culler, L. Liu, R. P. Martin, and C. Yoshikawa. LogP performance assessment of fast network interfaces. *IEEE Micro*, 1996.

[9] S. N. Damianakis. *Efficient Connection-Oriented Communication on High-Performance Networks*. PhD thesis, Dept. of Computer Science, Princeton University, May 1998. Available as technical report TR-582-98.

[10] C. Dubnicki, A. Bilas, Y. Chen, S. Damianakis, and K. Li. VMMC-2: efficient support for reliable, connection-oriented communication. In *Proceedings of Hot Interconnects*, Aug. 1997.

[11] T. Eicken, D. Culler, S. Goldstein, and K. Schauser. Active messages: A mechanism for integrated communication and computation. In *Proceedings of the 19th Annual Symposium on Computer Architecture*, pages 256–266, May 1992.

[12] R. Gillett, M. Collins, and D. Pimm. Overview of network memory channel for PCI. In *Proceedings of the IEEE Spring COMPCON '96*, Feb. 1996.

[13] V. Karamcheti and A. A. Chien. Software overhead in messaging layers: where does the time go? *ACM SIGPLAN Notices*, 29(11):51–60, Nov. 1994.

[14] K. Mackenzie, J. Kubiatowicz, M. Frank, W. Lee, V. Lee, A. Agarwal, and M. F. Kaashoek. Exploiting two-case delivery for fast protected messaging. In *The 4th IEEE Symposium on High-Performance Computer Architecture*, Feb 1998.

[15] A. M. Mainwaring, B. N. Chun, S. Schleimer, and D. S. Wilkerson. System area network mapping. In *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 116–126, Newport, Rhode Island, June 22–25, 1997. SIGACT/SIGARCH and EATCS.

[16] A. M. Mainwaring and D. E. Culler. Active Message applications interface and communication subsystem organization. Technical Report CSD-96-918, Computer Science Division, University of California at Berkeley, 1995.

[17] S. Pakin, M. Buchanan, M. Lauria, and A. Chien. The Fast Messages (FM) 2.0 streaming interface. Usenix'97, 1996.

[18] S. Pakin, M. Lauria, and A. Chien. High performance messaging on workstations: Illinois Fast Messages (FM) for myrinet. In *Supercomputing '95*, 1995.

[19] L. Prylli and B. Tourancheau. BIP: a new protocol designed for high performance. In *In PC-NOW Workshop, held in parallel with IPPS/SPDP98, Orlando, USA*, March 30 – April 3 1998.

[20] S. H. Rodrigues, T. E. Anderson, and D. E. Culler. High-performance local area communication with fast sockets. In *USENIX '97*, 1997.

[21] I. Schoinas and M. D. Hill. Address translation mechanisms in network interfaces. In *The 4th IEEE Symposium on High-Performance Computer Architecture*, 1998.

[22] H. Tezuka, A. Hori, and Y. Ishikawa. PM: a high-performance communication library for multi-user parallel environments. Technical Report TR-96015, Real World Computing Partnership, 1996.

[23] C. A. Thekkath, H. M. Levy, and E. D. Lazowska. Efficient support for multicomputing on atm networks. Technical Report 93-04-03, Department of Computer Science and Engineering, University of Washington, Apr. 1993.

[24] M. Welsh, A. Basu, and T. von Eicken. Atm and fast ethernet network interfaces for user-level communication. *Proceedings of the Third International Symposium on High Performance Computer Architecture (HPCA), San Antonio, Texas*, February 1997.

Author Biographies

Soichiro Araki received B.E. and M.E. degrees in Electrical Engineering from Kyoto University, Kyoto Japan, in 1987 and 1989, respectively. In 1989, he joined NEC Corporation's Opto-Electronics Research Laboratories, Kawasaki Japan. In 1995, he spent a year as a visiting researcher at NEC Research Institute in New Jersey, where he contributed to the analysis of communication performance in PC-clusters. He is currently researching a high-speed ATM switching system based on optical switching techniques in C&C Media Research Laboratories at NEC as an assistant manager.

Angelos Bilas received his B.S.E. degree in Computer Science from the Computer Engineering and Informatics Department, Patras University, Patras, Greece in 1993 and his M.A. degree in computer science from Princeton University in 1995. Currently, he is a Ph.D. student at Princeton University in the Department of Computer Science. His interests include distributed and parallel computing.

Cezary Dubnicki is a research staff member at Princeton University. His research interests are in parallel and distributed programming, including high-speed networks, distributed shared memory and high-performance message passing. He received an M.S. in computer science from Warsaw University and Ph.D. in computer science from the University of Rochester.

Jan Edler grew up in Maryland and went to the University of Maryland in College Park. After 3 years at Bell Labs in New Jersey, he joined the Ultracomputer project at New York University, where over the course of a decade he contributed to the design and implementation of 3 generations of multiprocessors and parallel operating systems. In 1994 he came to the NEC Research Institute, where he has been involved with PC-clusters, protein folding, user-mode thread systems, and cache simulation. He holds a Ph.D. from NYU.

Koichi Konishi received his M.S. degree in information engineering from the University of Tokyo in 1989, then joined C&C Systems Labs (now called C&C Media Labs), NEC Corporation. His research interests are in distributed and parallel programming.

James Philbin received his degrees (B.A., M.S., M.Phil., and Ph.D.) in Computer Science at Yale University. He is currently Director of Advanced Computing Projects at the NEC Research Institute. His research interests include programming languages and environments for parallel and distributed computing.