

Quantifying the Impact of Architectural Scaling on Communication

Taliver Heath, Samian Kaur, Richard P. Martin, Thu D. Nguyen
{taliver,rmartin,tdnguyen}@cs.rutgers.edu, samian@caip.rutgers.edu

Technical Report DCS-TR-418
Department of Computer Science
Rutgers University, Piscataway, NJ 08854-8019

Appears in Proceedings of the High Performance Computer Architecture Conference (HPCA), January 2001.

Abstract

This work quantifies how persistent increases in processor speed compared to I/O speed reduce the performance gap between specialized, high performance messaging layers and general purpose protocols such as TCP/IP and UDP/IP. The comparison is important because specialized layers sacrifice considerable system connectivity and robustness to obtain increased performance. We first quantify the scaling effects on small messages by measuring the LogP performance of two Active Message II layers, one running over a specialized VIA layer and the other over stock UDP as we scale the CPU and I/O components. We then predict future LogP performance by mapping the LogP model's network parameters, particularly overhead, into architectural components. Our projections show that the performance benefit afforded by specialized messaging for small messages will erode to a factor of 2 in the next 5 years. Our models further show that the performance differential between the two approaches will continue to erode without a radical restructuring of the I/O system. For long messages, we quantify the variable per-page instruction budget that a zero-copy messaging approach has for page table manipulations if it is to outperform a single-copy approach. Finally, we conclude with an examination of future I/O advances that would result in substantial improvements to messaging performance.

1. Introduction

This work considers the impact of expected architectural trends on messaging performance. In recent years, much research has focused on the design and implementation of specialized messaging systems [6, 36, 42, 45], reducing the fixed cost of sending and receiving messages to tens of instructions and a handful of bus operations. Increased performance, however, is typically gained only by trading off connectivity – the variety and number of potential entities a given messaging system can send to and re-

ceive from – and robustness. This trade-off is easy to understand in the context of parallel programming: connectivity and protocol robustness are secondary to performance because (i) the performance of many fine to medium-grained parallel programs is highly sensitive to communication performance [43], (ii) high connectivity is superfluous because processes of a parallel program only need to communicate with their peers running on the same system, and (iii) parallel programs and machines are carefully designed so that messages are lost only very rarely [7, 22]; a message loss is typically treated as a catastrophic event – either the program or the system will crash and need to be restarted. The need for a transport layer is thus eliminated by the programmer's fault model and the application requirements.

For the emerging class of large-scale distributed servers (e.g., web services), however, robustness and high connectivity are at least as important as performance. Three factors make protocol robustness critical: (i) these servers have very high availability requirements (e.g., minutes of downtime per year), implying that even occasional message loss cannot be catastrophic; (ii) intra-server communication depends on external client service demands, making it extremely difficult to exert enough control over the system “by design” to avoid message loss; and (iii) many commodity LANs do not implement sufficient hardware flow control to always prevent loss inside the network under arbitrarily adverse communication patterns.

High connectivity is also important because it allows system architects to select from a wide variety of hardware and software components, especially as such components evolve over time. It also allows designers to hedge technology risk, which is often ignored in a research setting, but is critical in an industrial context. For example, a messaging layer employing TCP/IP over Ethernet has extremely high connectivity, allowing the underlying cluster to be built using operating systems, network interfaces (NI), and switches from multiple vendors. On the other hand, a specialized messaging system that depends on a particular LAN with custom cards, switches, and protocols does not provide such high connectivity and so a site using this

technology is dependent on these technologies to last a long time and perhaps for a single vendor to be successful in the long run. While recent standards such as VIA [18] provide connectivity at the source code level, this is a far cry from the massive connectivity provided by the drivers, busses, media access control, and physical layers of general purpose networking. An example of the low connectivity of specialized messaging appears when trying to use VIA for a distributed Java-based server. The out-of-the-box Java runtime can not take advantage of this SAN layer without either building special software connectors or running general purpose messaging over it.

In this paper, we argue that current technology trends are bringing the performance of general purpose communication systems close enough to that of specialized systems such that designers of large-scale servers should seriously consider whether the trade off of connectivity and robustness justifies the added performance. Note that we are not advocating the abandonment of specialized messaging systems. Rather, we believe that the choice between specialized and general purpose needs to be considered carefully on a case-by-case basis. We break our analysis for this argument into two parts: one for small messages and the second for large messages.

For small messages, the performance of general purpose messaging systems has been steadily increasing while that of specialized layers has not. Figure 1 shows the LogP performance for small messages of four Active Message layers [11, 41, 43]. All these layers provide roughly the same interface, and each one provides isolation of messages between users. CMAM [43] ran on an MPP, while the others ran on workstation or PC hardware. Note that absolute performance has changed little since 1992 even though processor speed has roughly doubled each generation, going from 33, to 60, 167, and finally 400 MHz.

Intuitively, the reduction in the performance gap between general purpose and specialized messaging systems is easy to understand: the high cost of general purpose messaging arises from the large number of protocol instructions [28]. As processor speed increases, the cost of executing instructions drops correspondingly. Critically, however, *I/O devices have not kept pace with processor speed!* Consider that in 1992, a typical processor and I/O bus (e.g., a SPARC 2) both ran at 33 MHz. At these speeds, the cost of executing 2500 instructions dwarfs the cost of a few accesses over the I/O bus. Currently, on a 550 MHz Pentium III machine, this same 2500 instructions can be executed in less than twice the time required for same few I/O accesses. Architectural trends imply that we can expect this speed differential to be growing with time: processor performance doubles roughly every 2 years while I/O performance doubles every 4-7 years. Thus, Amdahl's law implies that the

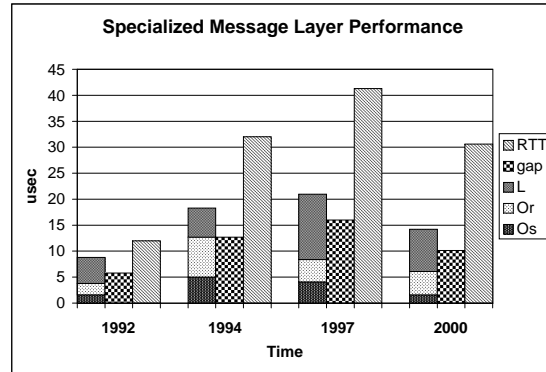


Figure 1. LogP performance of 4 Active Message Layers. The figure shows the LogP performance of 4 specialized message layers using the Active Message interface. The hardware spans 4 processor generations. The software layers are CMAM(1992), SSAM(1994), AMII(1997) and AMVIA(2000).

relative improvement gained by messaging systems which simply reduce instructions, as opposed to using more radical architectural innovations, will diminish with time.

In Section 3.1, we characterize the costs of sending and receiving small messages for two messaging systems using the LogGP model [3, 12]. Both systems export the AM-II interface [30]; one implements AM over a VIA LAN, the second implements AM on UDP/IP over Ethernet, giving the two layers significantly different levels of connectivity. We restrict ourselves to the AM-II interface because it contains many features needed by applications demanding more than just performance: thread support, blocking primitives, and error handling. We derive a simple cost model for mapping the LogGP parameters to two stock architectural components, the processor and the I/O interconnect; we also derive scaling rules for these components. We then use our model to study the effects of scaling on messaging costs over four speeds of Pentium-based PCs (233 MHz Pentium II to 550 MHz Pentium III). Next, we use the model to predict future communication costs.

Our results show that the performance differential between specialized and general purpose messaging systems has been decreasing steadily and will continue to decrease because of mismatching trends in processor and I/O speeds. Our projections show that a specialized messaging system will maintain a 2x performance advantage over general-purpose messaging. However, this is significantly less than the performance differential of 10x only six years ago [31].

In Section 4, we consider the cost of sending and receiving large messages. In particular, we consider archi-

tectural scaling effects on three messaging architectures: single-copy, zero-copy, and shared-memory communication. Single-copy is the simplest strategy and most current implementations of TCP and UDP use this approach. The second approach, zero-copy, manipulates the kernel page tables to allow sharing of user pages with the NI while maintaining normal copy semantics via copy-on-write. In shared-memory communication, the sender and receiver agree to transparently share a block of memory, whose mapping is supported by reverse page table manipulations on the NI.

Zero-copy is attractive compared to single-copy when the variable overhead of page-table manipulation is less than the per-page copy cost [40]. We refer to this cross-over point as the *OS page budget*. To examine the effects of architectural scaling, we estimate the number of instructions needed to setup a zero-copy transmission and compare this cost against copy costs as CPU speed and memory bandwidth scale through time. Our results show that copy reduction via software and architectural enhancements (e.g. checksum registers) remains important to obtaining high bandwidth. However, interestingly, our model shows that the OS page budget will scale only slowly with time.

2. Methodology

In this section, we first describe the LogGP model we use to characterize network performance. We then document our experimental apparatus and describe how we map LogGP parameters into architectural events, such as instruction count. We also document how we had to modify the interpretation of our microbenchmark results to fit our protocols into the LogGP framework.

2.1. The LogGP Model

When investigating communication architectures, it is important to recognize that the cost of each operation breaks down into portions that involve different resources: the processor, the memory, I/O busses, the network interface, and the actual network switches. However, it is also important that the communication cost model not be specific to particular hardware/software implementations. We use the LogGP model [3, 12] because it provides a middle ground by characterizing the performance of the key resources but not their structure.

LogGP characterizes a communication system using five parameters (Figure 2):

L: the *latency*, or delay, incurred in communicating a message containing a small number of words from its source processor/memory module to its target. The Latency includes the time spent in the network interfaces and fabric, but not in the processor.

o: the *overhead*, defined as the length of time that a processor is engaged in the transmission or reception of each message; during this time, the processor cannot perform other operations.

g: the *gap*, defined as the minimum time interval between consecutive message transmissions or consecutive message receptions at a module; this is the time it takes for a message to cross through the bandwidth bottleneck in the system.

G: the *Gap*, or time-per-byte for long messages. The inverse of *G* is the peak bandwidth. This parameter was added [3] because many platforms have special acceleration for long messages (e.g. DMA).

P: the number of processor modules.

2.2. Experimental Setup

We study two communication systems, AM-VIA and AM-UDP. Both systems export the AM-II API [30]. Figure 3 shows the LogGP parameters of the two systems. Both AM layers implement a three-way request/reply primitive for robustness, which UDP-AM also uses for correct handling of lost messages without requiring per-node buffering that grows with the number of processors [11, 31, 41]. Both AM layers implement flow control to avoid buffer overflows.

We measure LogGP parameters for AM-VIA and AM-UDP on four “generations” of PCs, including a 233 MHz Pentium II, a 300 MHz Pentium II, a 400 MHz Celeron, and a 550 MHz Pentium III. The 233–400 MHz machines have 66 MHz system buses while the 550 MHz machine has a 100 MHz system bus. All machines have 33 MHz PCI buses. For the AM-VIA measurements, we use a pair of Gigaset cLan cards connected back-to-back, running the clan-1.0.1 driver. For the AM-UDP measurements, we use a pair of Kingston 100 Mb/s ethernet cards (DEC DC21140 chips), running the tulip driver over a switched network. All experiments were run on Linux 2.2.12.

2.3. Characterizing Performance

After casting our communication systems into the LogGP model, we break down the LogGP parameters into their architectural costs. Our approach is to use the processor’s hardware event counters [24] to charge various hardware events to each parameter of the LogGP model. Specifically, we measure the following events:

- The number of instructions decoded.

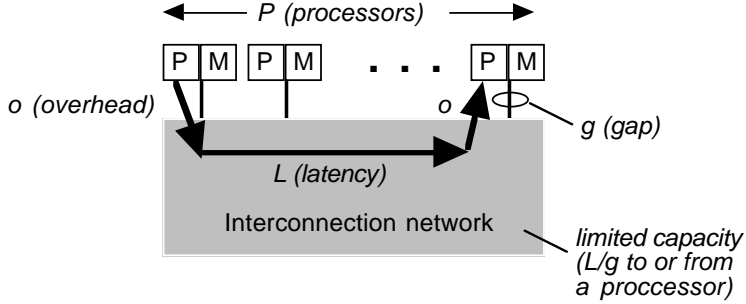


Figure 2. LogP. The LogP model describes an abstract machine configuration in terms of four performance parameters: L , the latency, o , the overhead, g , the gap between successive sends or successive receives, and P , the number of processor-memory modules.

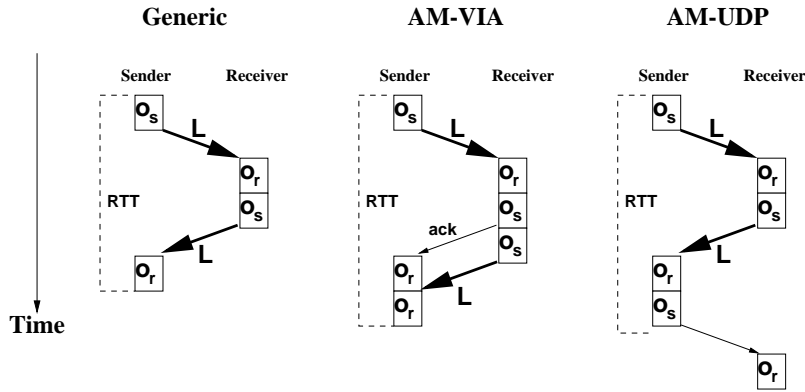


Figure 3. Protocols of 3 Active Message Layers. The internal protocols used by the “generic” layers in previous work and our two AM layers. The three-way protocols of our layers results in complex derivations of the LogGP parameters. We compute the send overhead, o_s , by direct measurement. We derive the receive overhead, o_r , by subtracting o_s from a steady-state sequence. We can compute L by computing the critical path of a round trip and subtracting o_s and o_r .

- The number of external bus accesses to memory space¹.
- The number of external bus accesses to I/O space – the x86 instruction set defines a 64K I/O space that is separate from memory space. Unfortunately, not all I/O operations are observable using this count as operations to memory mapped devices appear in memory space, not I/O space.
- The number of non-halted cycles.
- The number of interrupts.

We use the ubench benchmark [13] to measure the LogGP values for our communication systems. This benchmark first measures the send overhead, o_s by direct mea-

¹The memory bus is only critical when the cache is not large enough to hold the messages being transferred. For small messages, the cache was sufficient, and all memory bus transactions are considered to have moved across I/O bus as opposed to being memory accesses.

surement. Next, it measures gap, g . From these two parameters and knowledge of the protocol, we then infer the receive overhead, o_r . Finally, we can derive the latency, L , by measuring the round-trip-time (RTT) and subtracting out the overheads.

send overhead o_s is measured by sending a small burst of messages and computing the average cost per message. We charge events based on the average number of events per message measured during the burst. For example, if we send a burst of 4 messages and observe 32,000 instructions, the instruction count for o_s would be 800. A number of complications arose when we took this approach, however.

The first complication arose during the measurement of o_s for AM-UDP on the 550 MHz machine. When the tulip driver initiates a send, it quickly writes the send across the I/O bus and then immediately returns control to the user process. When the card receives this request and processes

it, it responds to the processor with an interrupt, at which time the remainder of the driver’s work to send the message is performed (i.e. cleaning up the send queue). A 550 MHz processor is fast enough that it often completes a small burst without ever getting an interrupt. This implies that the ethernet chipset used, the DEC/Intel 21140, overlaps L with part of o_s , reducing the accuracy of our LogGP model; a more accurate model would account for the initial overhead, o'_s , and the cost of the interrupt processing, o''_s . The effect of this architecture on our probing benchmark is that it lowers the observed o_s to o'_s , yet we must count o''_s to arrive at an accurate cost for o_s .

In order to model o_s with reasonable precision without expanding the model, we charge one interrupt for each o_s . To compute the interrupt cost, we measure the number of interrupts received in a burst and found the cost of sending when one interrupt was received per send and no messages has been received by the end of the burst.

A complication also arose for estimating the o_s for AM-VIA. On a slow machine the acknowledgement, shown in Figure 3 arrives before the burst is over. Thus, the ack processing time adds to the observed o_s . For example, Table 1 shows the measured number of I/O operations dropping with processor speed. However, the sum of o_s and o_r remains constant. In order to create a model of o_s independent of these effects we use the architectural event counts for o_s based on the measured 400 MHz machine results.

gap For a burst with only a small number of sends, the average messaging cost defines the send overhead, o_s . In bursts with large numbers of messages, the cost of each send approaches the steady-state initiation interval g . For both our communication systems, the bottleneck is the send and receive overheads. Figure 3 shows that in both our layers, an “extra” message is required in the three-way protocol. Taking into account this extra message, g is always equal to the steady-state cost of the slower side, which, assuming that $o_r > o_s$, is $o_s + 2o_r$.

receive overhead We can compute o_r in units of time by using measured values of o_s and g since $g = o_s + 2o_r$. To measure hardware events such as instructions executed, however, we only count events that are occurring on the sending side. Thus, for an event E , for AM-VIA, $E_{measured} = E_{o_s} + 2E_{o_r}$, while for AM-UDP, $E_{measured} = 2E_{o_s} + E_{o_r}$ (see Figure 3). Since we can measure E_{o_s} , we simply solve for E_{o_r} .

Again, we had to account for the tulip driver’s use of interrupt processing to reduce overhead on the 550 MHz machine. Furthermore, we also had to consider that the driver is able to consolidate work by servicing multiple sends in a single interrupt. For example, in steady state operation the driver is able to service two sends per interrupt on aver-

age. Of the two measured interrupts, we charge one to the receive and one to both sends.

Latency Measuring L with a 3-way protocol again requires care. We can see from Figure 3 that there is a *critical path* for a round-trip message. Assuming that other work is perfectly overlapped, we know the critical path of an RTT is composed of only o_s , o_r , and L . Having measured the RTT, o_s , and derived o_r , we can compute L for both protocols as: $RTT = 3o_s + 2o_r + 2L$. We only measure L in units of time since all events occur on the NIC and are unobservable from the CPU.

Gap To measure G , we send bursts of large messages, each with a fixed size. We then derive the bandwidth from the steady-state initiation interval and message size.

3. Small Message Scaling

In this section, we characterize the performance of our two messaging systems for small messages using the LogP model. We do not consider G as this parameter deals only with sending and receiving large messages.

3.1. Measured LogP Scaling

Figure 6 plots the measured LogP parameters of our two messaging systems against processor speed. Clearly, there are substantial performance advantages to using a specialized messaging system with hardware support, even when using a fairly complex protocol such as AM-II as the transport layer. When we examined the combined overhead of $o_s + o_r$, we found the ratio between the AM-VIA and the AM-UDP dropped from about 4.0 on the 233MHz machine to 3.3 on the 400 MHz machine. When we look at this same ratio for the 550 MHz machine, it jumps back up to 3.7, due to the increase of the memory bus speed.

To better understand why the performance differential between AM-VIA and AM-UDP is decreasing, we break o_s and o_r into their component costs, including instructions and I/O operations for both user and kernel space. We ignore L because previous work has shown that most applications can effectively overlap L with other computation [13]. Furthermore, Figure 6 shows that L is relatively constant for both AM-VIA and AM-UDP and so cannot be the cause of the decreasing performance differential. The measured L also serves as a check on our methodology. Figure 6 shows that, as expected, L stays relatively constant because the NIs and switches are the same throughout our experiments.

We characterize o_s and o_r in architectural terms using the following model:

$$\text{Time} = \frac{I_{CPU} \times CPI_{CPU}}{freq_{CPU}} + \frac{I_{I/O} \times CPI_{I/O}}{freq_{I/O}} \quad (1)$$

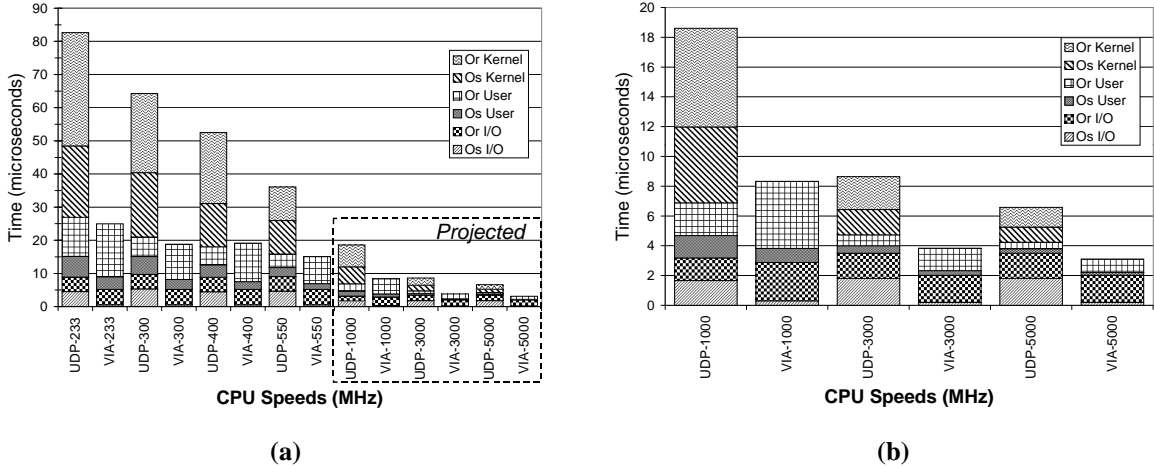


Figure 4. Measured and predicted LogP parameters as a function of processor speed. (a) Measured and predicted LogP values for processor speeds ranging from 233 MHz to 5 GHz; (b) Enlarge version of the predicted LogP values.

where $I_{I/O}$ is the number of I/O operations executed and $CPI_{I/O}$ is the average number of I/O bus cycles required to execute each operation in $I_{I/O}$, and I_{CPU} is the number of non-I/O instructions executed and CPI_{CPU} is the average number of processor cycles required to execute each instruction in I_{CPU} .

We measure the components in Equation 1 as follows. We use the Pentium performance counters to measure the number of cycles, instructions, memory accesses, and I/O operations as explained in Section 2. We assume that each I/O operation is generated by 1 instruction and $CPI_{I/O}$ equals 9 I/O bus cycles (based on the PCI specifications).

We note that our assumptions imply that multiple I/O operations are never overlapped or combined, making the actual cost of I/O potentially less than the model. We believe that this error is not a significant component compared to the divergence of CPU and I/O speeds.

Tables 1 and 2 give the measured instruction, cycle, and I/O counts for AM-UDP and AM-VIA respectively. Figure 6 plots the percentage breakdown of o_s and o_r . This figure shows that the fraction of time spent performing I/O operations is increasing. In the case of the 233 MHz processor, we see that AM-UDP spends approximately 11% of its time on these operations, and AM-VIA spends 21% of its time on similar pursuits. When we move to the 550 MHz platform, we see that the fraction of time spent on I/O operations has gone to 25% for AM-UDP, and 34% for AM-VIA. Correspondingly, the relative cost of executing protocol instructions vs. I/O operations is decreasing.

3.2. Predicted LogP Scaling

While protocol execution still accounts for a significant percentage of o_s and o_r on our fastest machine, the trend seems to suggest that, unless other architectural changes are made, I/O operations will eventually become the dominant factor in the performance of small messages. To follow this trend into the future, we derive scaling rules for the processor and I/O bus for the next five years as follows:

processor speed To predict processor speed, we extrapolate based on the past 7 years of performance data on x86 processors. Examining the clock rate and performance data from [32, 38], we can observe a rough rule of thumb of a 40% increase per year in clock rate and SPECint ratings. This level of improvement roughly corresponds to the “aggressive” predictions in [2]. At the time of this writing, 1 GHz processors are available. If processor speed doubles every 2 years (i.e., compounding at 40% year), then we would expect 3 GHz processors to be available by summer 2003 and 5 GHz by 2005. We assume that architectural enhancements, such as larger caches and microarchitecture improvements, will keep CPI for messaging at roughly the same levels as today.

protocol instructions Barring a revolution in OS design or messaging software, the number of instructions should remain relatively constant. We do not model any drop in the number of instructions.

I/O bus speed The growth trend of I/O bus speed has been considerably different from that of CPU speed, demonstrat-

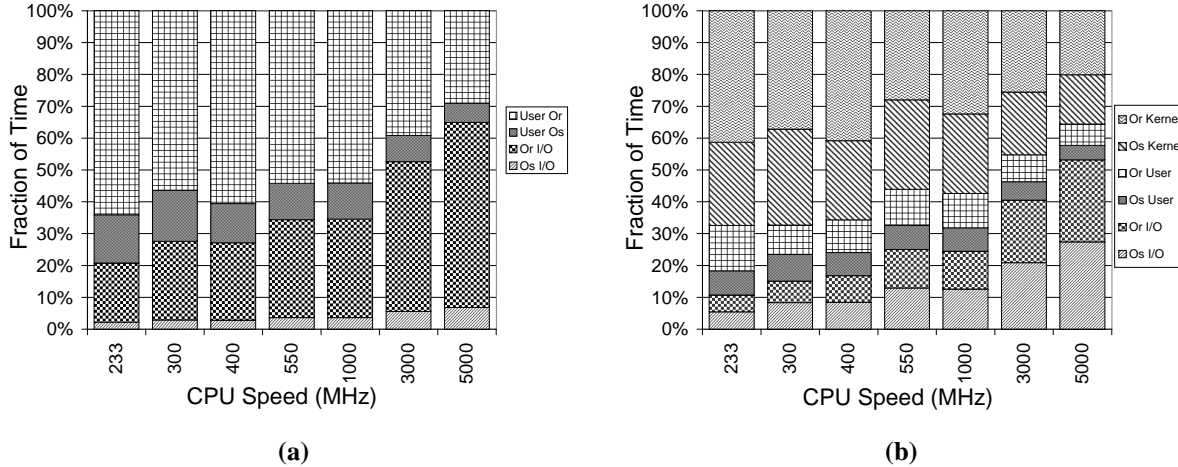


Figure 5. Predicted breakdown of o_s and o_r . Percentage cost of components of o_s and o_r for (a) AM-VIA and (b) AM-UDP as a function of processor speed.

ing increases every 4 to 7 years (e.g., ISA in 1984, EISA in 1988, 33 Mhz PCI in 1993, 66 MHz PCI in 2000). One factor that dampens the growth of bus speed is the fact that the number of slots available per I/O bridge tends to decrease as bus speed increases, increasing the cost of a fixed number of I/O slots. For example, at 133MHz, each PCI bus can support at most one PCI slot, requiring multiple PCI busses for more slots. We thus believe that the standard bus speed for PCs over the next five years will remain at 66 MHz.

I/O operations In addition to scaling I/O bus speed, we must also consider the CPI of I/O operations as the bus width increases. The current 33 MHz PCI bus on our machines is 32-bit wide and requires 9 cycles per operation between the CPU and a PCI card. In the future, we expect the bus width to double to 64 bits, but the number of cycles to transmit a given instruction will remain the same. However, a driver may batch multiple I/O operations to take advantage of the wider bus. Such batching will vary with card architecture as well as driver implementation. To account for this potential optimization, we adjust the number of bus clock cycles per I/O operation from 9 to 7.

Figure 4 and 5 give the predicted scaling of o_s and o_r and their components costs. Figure 4 shows that the performance gap between specialized and general purpose messaging systems will continue to decrease, dropping from a ratio of 3.30 (for the sum of o_s and o_r) at 233 MHz to 2.39 at 550 MHz to 2.12 at 5 GHz². Figure 5 shows that

²If we take the optimistic assumption that bus speed will be at 133 MHz in five years, the ratio will spike up from 2.25 to 2.30 when processor reaches 3 GHz to 2.12 when processor speed reaches 5 GHz. After the spike, however, the ratio will continue to decrease once again.

this decrease in the performance gap is due to the increasing importance of I/O operations. At processor speed of 5 GHz, I/O operations account for 53% of the overhead for AM-UDP and for 65% of the overhead for AM-VIA.

If this trend continues beyond our projection period of 5 years, then only a difference in the number of I/O instructions (and the corresponding CPI) will produce a significant difference in performance; the amount of time necessary to perform the user and kernel instructions will become insignificant.

4. Large Messages

In this section we examine the effects of architectural scaling on large messages, focusing on the per-byte overhead. In particular, we investigate the architectural scaling of the cross-over point between the cost of copying a page vs. using memory management techniques to share the page between the user process and the NI.

Broadly speaking, three approaches to high bandwidth messaging have evolved, two maintaining the classical messaging API, where the user process is free to modify the message buffer once the `send` primitive returns (e.g., as in socket and MPI), while the third exports a significantly different API. The first, and simplest, is to use a staging area for both in-bound and out-bound data, requiring each message to be copied at least once. We call this the *single-copy* approach because most implementations of general purpose messaging layers have been able to achieve this lower limit. Although single-copy approaches can deliver good performance [15], it increases the load on the memory bus and can also increase the per-byte overhead over the next two

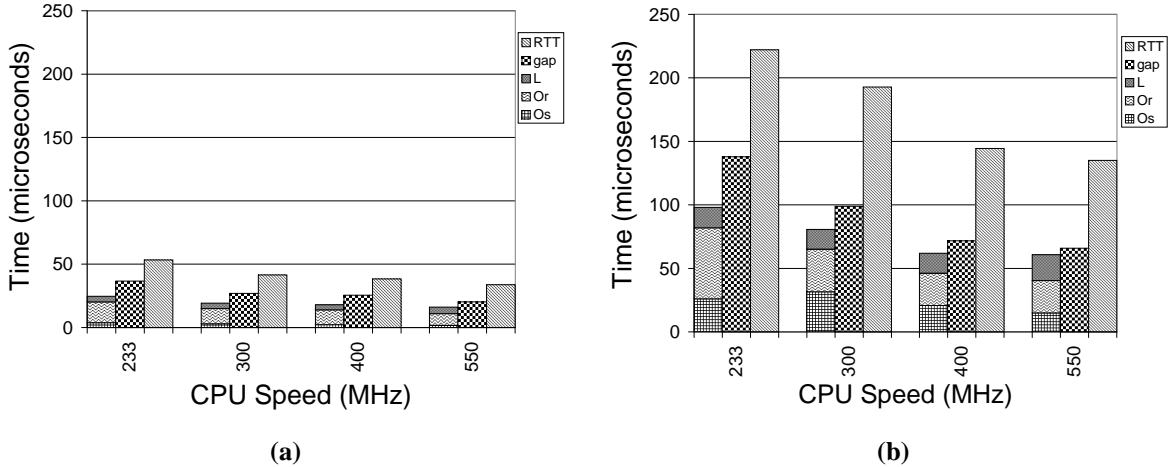


Figure 6. Measured LogP parameters as a function of processor speed. *LogP* values for (a) AM-VIA and (b) AM-UDP as a function of processor speed.

approaches.

The second approach, called *zero-copy messaging*, manipulates the kernel page tables to allow sharing of user pages with the NI while maintaining the same semantics as single-copy through copy-on-write. Although this approach has been studied extensively [8, 9, 29], it has not been mapped into architectural parameters. In particular, page table manipulation, the fundamental mechanism used in zero-copy, is complex. Thus, zero-copy only becomes attractive compared to single-copy when the per-page overhead of page-table manipulations is less than the per-page copy cost [40]. We call this cross-over point the *OS page budget*.

The final approach is *shared-memory* communication [6, 11, 20, 34]. The basic idea behind this approach is that the sender and the receiver agree to transparently share a block of memory. Communication then occurs via writes, DMA, or sometimes reads, into the shared region. While shared-memory approaches deliver excellent performance, they have several drawbacks. The primary drawback is that classical messaging (e.g., sockets and MPI) does not map well into a pure shared-memory approach; constructing a scalable, N-to-1 queue is difficult without hardware memory coherence or specialized support such as fetch-and-add registers. A more subtle problem with shared-memory communication is the considerable loss of connectivity because of the “commonality” required between the OS and NI of both the sender and receiver.

A second cost of shared-memory communication over zero-copy is the cost of mapping shared page tables between the NI and CPU. These page tables require additional communication between the NI and CPU that is not present in

a zero-copy system. While many studies have documented messaging performance once all the shared mappings are in place, few have reported these set-up costs. The amortization of the mappings depends on the application and higher-level libraries, which we do not pursue in this work.

4.1. Implementing Single-Copy

Figure 8 shows the per-byte cost of our two messaging systems as measured on the 400 MHz Celeron machines. From this data, we compute that AM-VIA requires approximately 1.65 cycles per byte to send large messages. Interestingly, the per-byte cost for AM-UDP is much higher than that for AM-VIA. To understand why, we breakdown the AM-UDP per-byte cost into its user-level and kernel-level components. This breakdown shows that AM-UDP is making two copies, one at the user level (in the AM layer) and one in the kernel (in the UDP/IP protocol stack). We attribute the slightly higher kernel-level per-byte cost to checksumming.

The fact that two copies are required in AM-UDP points to a deficiency in its architecture. Because the AM and UDP layers are not “aware” of each other, each makes a copy independently. This is a well-known problem with layering (e.g., [17]) but, in this context, points to the potential performance disadvantage of splitting a messaging layer across the user-kernel boundary where there is significant functionality on both sides.

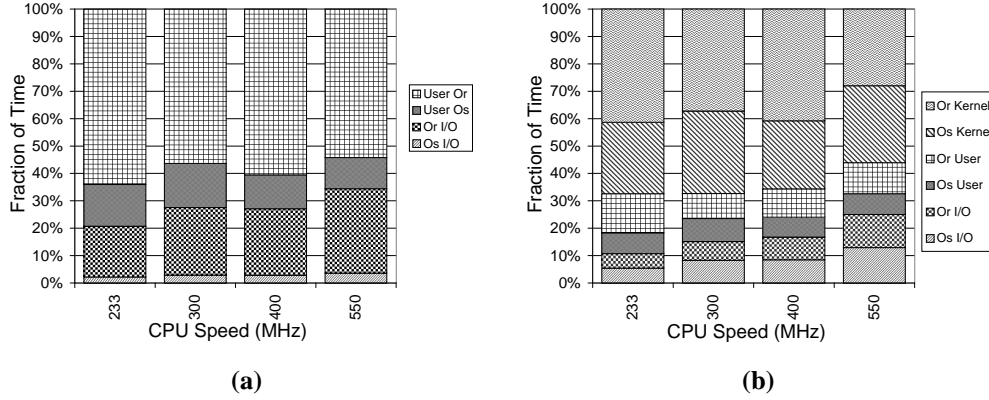


Figure 7. Break down of o_s and o_r . Percentage cost of components of o_s and o_r for (a) AM-VIA and (b) AM-UDP as a function of processor speed.

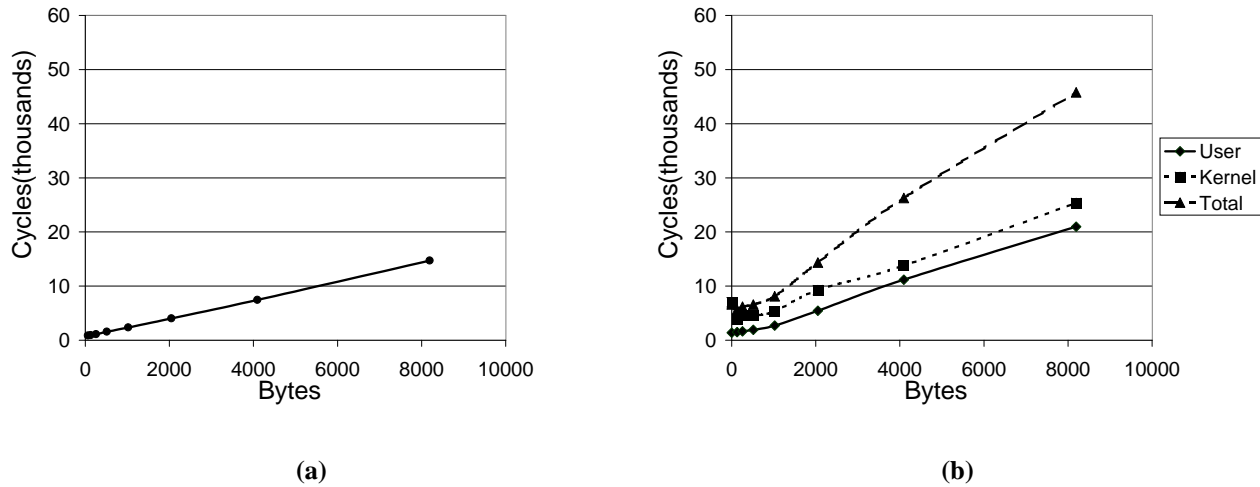


Figure 8. Per-byte overhead. Cycle cost to send a message for (a) AM-VIA and (b) AM-UDP as a function of message size.

4.2. Page Table Operations

While zero-copy and shared-memory communication can avoid data copying, a known performance bottleneck, the required page table manipulations are complex. For example, on the send side, the OS must typically:

1. Check that the region to send is valid.
2. Fault in any paged-out pages.
3. Pin the pages, i.e, marking them as un-swappable, if the region is swappable.
4. Change the access permission of each page to read-only.
5. Translate the page-addresses into the correct physical or bus addresses needed by the NI.

6. Un-pin the pages once I/O is complete.
7. Change the access permission of each page back to the original permission.

Note that this sequence does not implement copy-on-write, but allows for it on a write-fault. To actually implement this would require even more complexity.

The receive side is somewhat easier than the send-side. Here the OS must:

1. Un-pin the pages, i.e, marking them as swappable, if the user's region is not pinned.
2. Change the access permission of each page to writable if required.
3. Change the user's page-table entries for the receive region to point to the received pages.

CPU MHz	Burst Size	Instructions	Cycles	I/O Space	Memory Space
233	4	630 ±6	895 ±11	8.01 ±0.02	0.0 ±0.0
	128	5326 ±8	6277 ±21	29.7 ±0.05	0.0 ±0.0
550	4	630 ±2	920 ±4	2.25 ±0.013	0.0 ±0.0
	128	5401 ±12	6933 ±17	36 ±0.5	0.0 ±0.0

Table 1. Measured messaging costs in AM-VIA. Measured number of user instructions, kernel instructions, cycles, and I/O operations for AM-VIA. The average number of operations are shown for a short burst size of 4 messages and a long burst size of 128 messages. The reported event counts do not include any of the adjustments described in Section 2. The 300 and 400 MHz machine results have been omitted due to space constraints.

	CPU MHz	Burst Size	Instructions	Cycles	I/O Space	Memory Space
User	233	4	829 ±37	1460 ±12	0 ±0	0 ±0
		128	3409 ±18	6963 ±33	0 ±0	0 ±0
	550	4	806 ±3.4	1378 ±18	0 ±0	0 ±0
		128	3420 ±25	7273 ±48	0 ±0	0 ±0
Kernel	233	4	2781 ±32	6068 ±137	11.0 ±0.1	5.5 ±0.2
		128	11574 ±34	25172 ±36	30.2 ±0.1	14.1 ±0.1
	550	4	1696 ±73	4456 ±222	1.0 ±0.0	4.6 ±0.2
		128	11398 ±48	28308 ±185	21.5 ±0.1	14.4 ±0.1
Total	233	4	3555 ±24	7504 ±112	11.0 ±0.1	5.5 ±0.2
		128	15243 ±42	32117 ±55	29.3 ±0.1	14.3 ±0.1
	550	4	2411 ±46	4764 ±237	1.0 ±0.0	4.0 ±0.1
		128	14611 ±72	34687 ±80	22.0 ±0.2	14.6 ±0.1

Table 2. Measured messaging costs in AM-UDP. Measured number of user instructions, kernel instructions, cycles, and I/O operations for AM-UDP. The average number of operations are shown for a short burst size of 4 messages and a long burst size of 128 messages. The reported event counts do not include any of the adjustments described in Section 2. As above, the 300 and 400 MHz machine results have been omitted due to space constraints.

4. Return the old pages back to the kernel or device.

A careful study of the true cost of these operations crosses far into the operating systems field and is beyond the scope of this work. Our contribution instead is to quantify how the OS page budget will scale with time. To give the reader some context of the cost of the above OS operations, we measured the cost of the VIA call to register memory for the Giganet driver, the approximate cost of user-level page-table manipulations via a pair of `mlock()/munlock()` and two `mprotect()`, and copy cost via `bcopy()`. Figure 9 show the results on a 400 MHz Celeron machine.

We make two observations. First, it may be less costly to use a single-copy approach when the message size is one page or less³. Second, the cost of VIA memory registration is very close to basic page table manipulation costs for small buffer sizes (less than 64KB) – about 10us for

³Since we’re only estimating the cost of zero-copy, we cannot say definitively where this cross-over point currently is.

4K page size – but degrades to about 3 cycles per byte, or 133MB/s for buffer sizes above 64KBs. These results imply that on-demand pinning and un-pinning of small regions has favorable costs, but for large messages, applications should resort to managing communication with a fixed cost pinned buffer. We are currently investigating the Giganet driver to fully understand why registering 64K regions is as fast as basic page table manipulations but approaches copying cost for larger regions.

4.3. Predicting the OS page budget

We use the following simple model to examine how memory and processor scaling affects the cost of sending/receiving large messages via a single-copy vs. a zero-copy architecture:

$$t_{copy} = \frac{page_size}{memory_bandwidth} \quad (2)$$

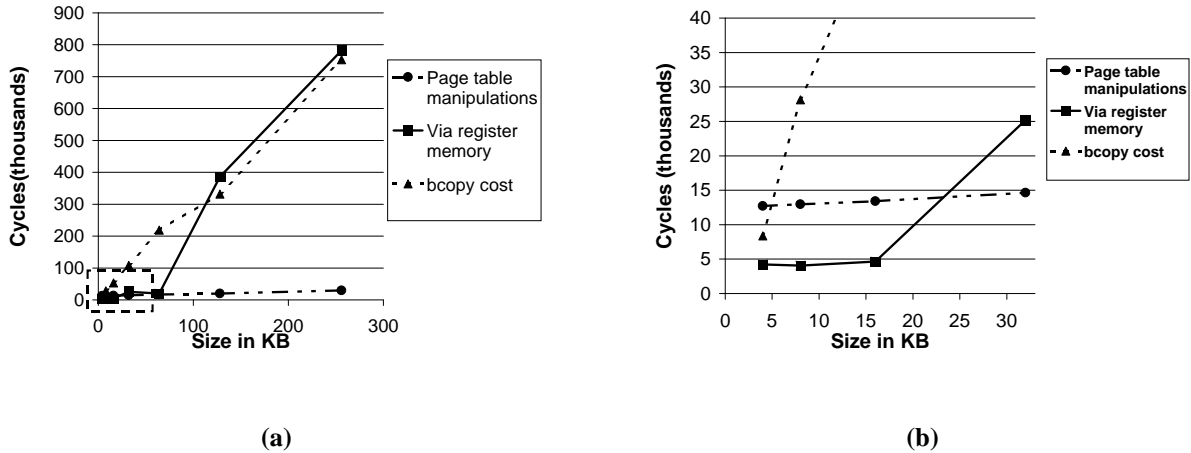


Figure 9. Per-byte cost of copying, page table manipulations, and memory registration. (a) Cycle cost for copying, page table manipulations, and memory registration vs. message size; (b) Blow-up of (a) for message sizes between 4 and 30 KB.

$$t_{ptm} = \frac{num_instructions \times CPI}{processor_speed} \quad (3)$$

where t_{copy} is per-page copy time under single-copy and t_{ptm} is the per-page setup time under zero-copy. (Note that both t_{copy} and t_{ptm} are per-page variable costs.) The OS page budget is computed by setting t_{copy} equal to t_{ptm} and solving for $num_instructions$. We assume a CPI of 2.0, which is consistent with the CPI’s observed across a range of processor speeds in Section 3.1.

We use the same processor speed vs. time predictions that we derived in Section 3.1. To derive predictions for memory bandwidth, we use data available from the STREAM benchmark site [32], which shows that memory bandwidth has been increasing at a rate of roughly 35% per year. Figure 10(a) shows the relationship of clock speed to memory bandwidth. The linearity of the curve may be surprising but is consistent with the fact that both processor speed and memory bandwidth are growing exponentially.

Figure 10(b) shows the OS page budget in terms of number of instructions as the processor scales to 5 GHz. An interesting effect of our scaling rule is that, if the page size remains fixed at 4 KB as the processor and memory get faster, the OS page budget approaches an asymptotic limit of 10240 instructions. Intuitively, because the copy cost for a fixed-size page decreases as an exponential function, the limit of the exponential decay results in a fixed page budget. This asymptotic limit is directly related to the page size; doubling the page size doubles this limit. Thus, as page size increase, it becomes ever more advantageous to use a zero-copy approach.

4.4. Zero-copy and Checksumming

Most general-purpose protocols require or are run with higher-level checksums compared to specialized protocols which only run with a layer-2 checksum. Our OS page budget computed above does not include a checksum cost. If checksumming is to be supported, a single-copy approach become more attractive because the copy and checksum can be amortized during the same operation. An alternative for zero-copy approaches is to provide support on the NI for computing checksums.

5. Related Work

The last 10 years has seen tremendous efforts investigating high performance messaging layers [6, 36, 42, 43]. In addition, several projects describe methods of providing a measure of classic abstractions on top of these layers [16, 37]. Many of these projects have provided detailed analysis or performance models [4, 13, 26]. However, these models were always in the context of specialized communication systems, thus making comparisons to general purpose systems difficult.

In a more general networking context, much work has been done to quantify the performance of existing IP protocol stacks [27, 28], or increase their performance via copy reduction techniques [8, 9, 15, 29, 40].

Unlike the analysis of specialized messaging, the performance of these general purpose systems is rarely defined in terms of architectural models. It is difficult from these studies to determine the effect of the processor, memory or I/O

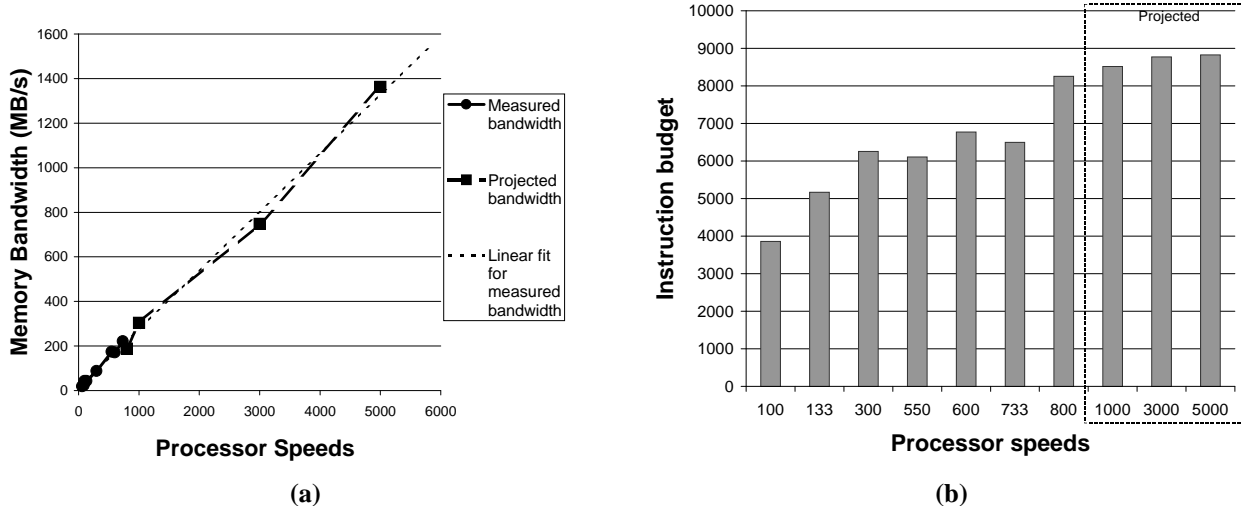


Figure 10. Predicted OS page budget. (a) Measured and predicted memory bandwidth as a function of processor speed; (b) Measured and predicted OS page budget as a function of processor speed.

systems' impact on communication performance. The same situation exists for popular micro-benchmarks for measuring the performance of these layers [25, 33, 10].

The effect of different network card organizations was investigated in [39]. However, that work did not extend projections as to which organization would best track architectural trends.

Perhaps the closest work to our own can be found in [8]. The linear models provided in the paper were scaled in a qualitative way with processor and memory speeds. However, architectural characteristics, such as the number of instructions or cycles, were not given.

6. Future I/O Architectures

Current industry efforts to improve I/O architecture focuses on delivering increased bandwidth. For example, the Infiniband [23] is a complex specification that will likely increase latency over a stock PCI bus. However, Infiniband's point-to-point switching will increase total deliverable bandwidth. In addition, its complex packet format is designed to increase connectivity by allowing disks, computers and network interfaces to share a common fabric.

On the research side, many I/O enhancements have been proposed or prototyped. Most of these take the approach of integrating the messaging unit closer to the processor. While all these approaches reduce latency and increase bandwidth, they decrease the overall connectivity, sometimes by substantial amounts. These approaches can be

summarized as:

- Integrate messaging into the processor [14].
- Integrate messaging into the cache-controller [1, 21].
- Integrate the messaging unit into the memory system [35]. A more radical approach integrates it into the DRAM slots [34].

The key challenge for future high-performance networking will be to maintain a high degree of connectivity while increasing performance. This means either working within the confines of the kernel and I/O busses or completely replacing the standards which form the underlying communication substructure. Recent work in operating systems [5, 19] has been able to achieve high-performance networking in a kernel context by using experimental operating systems. Such systems, however, achieve their performance by reducing software connectivity.

Perhaps the best example of a system which provides both high connectivity and performance is [9]. In that work, a few simple hardware and software modifications resulted in a large increase in the deliverable bandwidth in FreeBSD. The hardware techniques included large MTUs and checksum offload. The primary modification to the OS was to provide zero-copy send and receive. The results showed a factor of 30% bandwidth improvement without an appreciable increase in latency.

In the near future the best way to achieve high-performance and connectivity will be for network standards

bodies to adopt large MTUs, for card vendors to provide checksum offloading and implement large MTUs, and for OS vendors to incorporate driver API's that allow checksum offload and zero-copy techniques. These techniques alone will allow applications to use a substantial portion of the hardware bandwidth in the context of the operating system. Latency reduction will come as processor performance outstrips the rest of the I/O system. These approaches, unlike specialized messaging, however, require a common set of standards to be accepted by four distinct communities: the operating system, network interface, switch and motherboard vendors. The speed and adoption of new standards may become the limiting factor for I/O performance rather than any technological hurdles.

In the longer term, the real challenge will be to reduce general purpose messaging overheads. On the hardware side, better integration into the memory system and techniques such as cacheable control registers [44], may result in systems which can eliminate the I/O bus.

Reducing the LogGP latency, as distinct from overhead, will remain difficult however. As technology scales, the cost of moving data through each chip will continue to drop, but as long as systems are comprised of discrete components, there will be significant latency costs. Programmers and OS designers should thus continue to focus on algorithms which tolerate latency.

7. Conclusions

For small messages, we have shown how the relative improvements in the processor and I/O bus result in an erosion in the performance differential between specialized and general purpose messaging systems. We have observed a reduction of this difference from a factor of 10 four years ago to a factor of 4 today. The reason for this reduction is that I/O bus speeds have not kept up with processor improvements. While processor performance doubles every 2 years, I/O busses only increase in performance every 4-7 years.

Extending our models out five years, we predict specialized layers to have software overheads of 2-3 times better than general purpose systems. This should give system designers pause as to whether such performance increases are worth the loss of connectivity with its attendant risks of technology abandonment.

For long messages, we quantified that the breakpoint between copying and zero-copy page pinning will be approximately 10,000 instructions per 4KB page and double that for 8KB pages. This will give OS designers a clear target for implementing zero-copy protocols in the future. Measuring the cost of basic page manipulations in the Linux kernel showed that these costs are not unreasonable. However, sufficient hardware support, such as checksum registers, will

be required to make zero-copy protocol stacks viable.

Acknowledgements

We would like to thank Ricardo Bianchini for his comments, DiscoLab for putting up with the inconvenience of disassembling each machine, and the anonymous reviewers for their comments. We also thank Brent Chun providing the UDPAM source code. This work is supported by NSF-9986046.

References

- [1] A. Agarwal, R. Bianchini, D. Chaiken, K. Johnson, D. Kranz, J. Kubiawicz, B. H. Lim, K. Mackenzie, and D. Yeung. The MIT Alewife Machine: Architecture and Performance. In *Proceedings of the 22nd International Symposium on Computer Architecture*, May 1995.
- [2] V. Agarwal, H. S. Murukkathampoondi, S. W. Keckler, and D. C. Burger. Clock Rate Versus IPC: The End of the Road for Conventional Microarchitectures. In *Proceedings of the 27th International Symposium on Computer Architecture*, May 2000.
- [3] A. Alexandrov, M. Ionescu, K. E. Schauer, and C. Scheiman. LogGP: Incorporating Long Messages into the LogP model - One step closer towards a realistic model for parallel computation. In *7th Annual Symposium on Parallel Algorithms and Architectures*, May 1995.
- [4] S. Araki, A. Bilas, C. Dubnicki, J. Edler, K. Konishi, and J. Philbin. User-Space Communication: A Quantitative Study. In *Proceedings of the High Performance Networking and Computing (SC98)*, Orlando, FL, Nov. 1998.
- [5] B. N. Bershad, C. Chambers, S. Eggers, C. Maeda, D. McNamee, P. Pardyak, S. Savage, and E. G. Sier. SPIN—An Extensible Microkernel for Application-Specific Operating System Services. Technical report, University of Washington, 1994.
- [6] M. A. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. Felten, and J. Sandberg. Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer. In *Proceedings of the 21st International Symposium on Computer Architecture*, Apr. 1994.
- [7] 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):29–38, Feb. 1995.
- [8] J. C. Brustoloni and P. Steenkiste. Effects of Buffering Semantics on I/O Performance. In *Proceedings of the 2nd USENIX Symposium on Operating Systems Design and Implementation*, Oct. 1996.

- [9] J. S. Chase, K. G. Yocum, and A. J. Galatin. End-System Optimizations for High-Speed TCP. Submitted For Publication. Also see <http://www.cs.duke.edu/ari/publications/>, June 2000.
- [10] Chesapeake Computer Consultants, Inc. *Test TCP (TTCP)*, 1997. <http://www.ccci.com/tools/ttcp/>.
- [11] B. N. Chun, A. M. Mainwaring, and D. E. Culler. Virtual Network Transport Protocols for Myrinet. *IEEE Micro*, 18(1):53–63, 1998.
- [12] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauer, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 262–273, 1993.
- [13] D. E. Culler, L. T. Liu, R. P. Martin, and C. O. Yoshikawa. Assessing Fast Network Interfaces. In *IEEE Micro*, volume 16, pages 35–43, Feb. 1996.
- [14] W. J. Dally, J. S. Keen, and M. D. Noakes. The J-Machine Architecture and Evaluation. In *COMPCON*, pages 183–188, Feb. 1993.
- [15] C. Dalton, G. Watson, D. Banks, and C. Calamvokis. Afterburner (network-independent card for protocols). *IEEE Network*, 3(4):36–43, July 1993.
- [16] S. Damianakis, C. Dubnicki, and E. W. Felten. Stream Sockets on SHRIMP. In *Workshop on Communication and Architectural Support for Network-based Parallel Computing*, Feb. 1997.
- [17] P. Druschel and L. L. Peterson. Fbufs: A High-Bandwidth Cross-Domain Transfer Facility. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pages 182–202, Dec. 1993.
- [18] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, F. Berry, A. M. Merritt, E. Gronke, and C. Dodd. The Virtual Interface Architecture. *IEEE Micro*, 18(2):66–76, Mar. 1998.
- [19] D. R. Engler, M. F. Kaashoek, and J. J. O’Toole. Exokernel: An Operating System Architecture for Application-Level Resource Management. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, Dec. 1995.
- [20] R. B. Gillett. Memory Channel Network for PCI. In *IEEE Micro*, volume 16, pages 12–18, Feb. 1996.
- [21] M. Heinrich, J. Kuskin, D. Ofelt, J. Heinlein, J. Baxter, J. P. Singh, R. Simoni, K. Gharachorloo, D. Nakahira, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy. The Performance Impact of Flexibility in the Stanford FLASH Multiprocessor. In *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 274–285, Oct. 1994.
- [22] R. Horst. TNet: A Reliable System Area Network. *IEEE Micro*, 15(1):37–45, Feb. 1995.
- [23] Infiniband trade Association. *Infiniband Trade Association Home Page*. <http://www.infiniband.org>.
- [24] Intel Corporation, Santa Clara, CA. *Pentium Pro family developer’s manual, volume 3: Operating system writer’s manual*, 1996. Order number 242692.
- [25] R. Jones. Netperf 2.1 Homepage. <http://www.cup.hp.com/netperf/NetperfPage.html>, Feb. 1995.
- [26] V. Karamcheti and A. Chien. Software Overhead in Messaging Layers: Where Does the Time Go? In *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 1994.
- [27] J. Kay and J. Pasquale. The Importance of Non-Data-Touching Overheads in TCP/IP. In *Proceedings of the 1993 SIGCOMM*, pages 259–268, San Francisco, CA, September 1993.
- [28] K. Keeton, D. A. Patterson, and T. E. Anderson. LogP Quantified: The Case for Low-Overhead Local Area Networks. In *Hot Interconnects 3*, Stanford University, Stanford, CA, August 1995.
- [29] H. keng Jerry Chu. Zero-Copy TCP in Solaris. In *Proceedings of the 1996 USENIX Conference*, Jan. 1996.
- [30] A. Mainwaring. Active Message Applications Programming Interface and Communication Subsystem Organization. Technical Report CSD-96-918, University of California, Berkeley, Computer Science Division, Oct. 1996.
- [31] R. P. Martin. HPAM: An Active Message Layer for a Network of Workstations. In *Proceedings of Hot Interconnects 2*, July 1994.
- [32] J. D. McCalpin. Sustainable Memory Bandwidth in Current High Performance Computers. <http://home.austin.rr.com/mccalpin/papers/bandwidth/bandwidth.html>, Oct. 1995.
- [33] L. McVoy and C. Staelin. Imbench: Portable Tools for Performance Analysis. In *Proceedings of the 1996 USENIX Conference*, Jan. 1996.
- [34] R. Minnich, D. Burns, and F. Hady. The Memory Integrated Network Interface. *IEEE Micro*, Feb. 1995.
- [35] S. S. Mukherjee and M. D. Hill. Making Network Interfaces Less Peripheral. *IEEE Computer Magazine*, 31(10):70–76, Oct. 1998.
- [36] S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet. In *Supercomputing ’95*, San Diego, California, 1995.
- [37] S. H. Rodrigues, T. E. Anderson, and D. E. Culler. High-Performance Local-Area Communication Using Fast Sockets. In *Proceedings of the 1997 USENIX Conference*, Anaheim, CA, Jan. 1997.
- [38] Standard Performance Evaluation Corp. *SPEC CPU 95 Benchmarks*, 1995. <http://www.specbench.org/osg/cpu95>.

- [39] P. Steenkiste. A Systematic Approach to Host Interface Design for High-Speed Networks. *IEEE Computer*, 18(1):53–63, 1998.
- [40] M. N. Thadani and Y. A. Khalidi. An Efficient Zero-Copy Framework for Unix. Technical Report SMLI TR-95-39, Sun Microsystems Laboratories, May 1995.
- [41] T. von Eicken, V. Avula, A. Basu, and V. Buch. Low-Latency Communication over ATM Networks using Active Messages. In *Proceedings of Hot Interconnects 2*, Stanford University, Stanford, CA, July 1994.
- [42] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In *Proceedings of the Fifteenth SOSP*, pages 40–53, Copper Mountain, CO, December 1995.
- [43] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauer. Active Messages: a Mechanism for Integrated Communication and Computation. In *Proc. of the 19th Int'l Symposium on Computer Architecture*, May 1992.
- [44] D. A. Wood, S. K. Reinhardt, S. S. Mukherjee, B. Falsafi, M. D. Hill, and R. W. Pfile. Cacheable interface control registers for high speed data transfer. United States Patent 5,951,657.
- [45] K. G. Yocum, J. S. Chase, A. J. Gallatin, and A. R. Lebeck. Cut-Through Delivery in Trapeze: An Exercise in Low Latency Messaging. In *Symposium on High-Performance Distributed Computing (HPDC)*, Portland, OR, Aug. 1997.