In Proceedings of Parallel Computer Routing and Communications Workshop,
Seattle, Washington, May 16-18, 1994.

Do Faster Routers Imply Faster Communication?

Vijay Karamcheti and Andrew A. Chien

Department of Computer Science
University of Illinois at Urbana-Champaign
1304 W. Springfield Avenue, Urbana, IL 61801
{*vijayk,achien*}*@cs.uiuc.edu*

Abstract. Despite significant improvements in network interfaces and software messaging layers, software communication overhead still dominates the hardware routing cost in most parallel systems. In this study, we identify the sources of this overhead by relating user communication services to particular network hardware features. Based on a detailed analysis of the *active messages* layer on the CM-5, we assign the software messaging cost to specific user communication services and network features. Our study shows that 50–70% of the software cost of messaging can be attributed to providing end-to-end flow control, in-order delivery, and reliable transmission services. This overhead is a direct effect of specific network features – arbitrary delivery order, finite buffering, and limited fault-handling – and is unlikely to be eliminated through improved software implementations. We conclude that reducing this software overhead requires changing the constraints on messaging layers: we propose designing routing networks which replace (or make unnecessary) software to implement user services. Specifically, networks which provide ordering, end-to-end flow control, and hardware-supported fault-tolerance can significantly reduce the end-to-end cost of communication.

1 Introduction

In highly parallel machines, a collection of computing nodes work in concert to solve large application problems. The nodes communicate data and coordinate their efforts by sending and receiving messages through the underlying communication network. Thus, the performance achieved by such machines critically depends on the end-to-end cost of communication mechanisms. There are several contributors to this cost: the routing time, the time to get messages into and out of the network, and software protocol overhead. These times are determined by the design of the routing network and processor-network interface as well as the choice of software protocols.

Recent advances in messaging implementations [25] and improved network interfaces have reduced the software cost of messaging significantly. Despite this progress, software communication overhead still dominates the hardware routing cost. Our study focuses on understanding the relationship between user communication requirements, messaging layer components, and particular hardware features. Messaging layers act as a bridge, enhancing the functionality of the raw hardware to provide higher-level communication abstractions required by user application programs. In this paper, we precisely characterize the software costs in a prototypical messaging layer. This characterization can be viewed as the cost of messaging layer services, or alternately as the cost of particular network hardware features which necessitate software to support those services.

We use the *active messages* layer (CMAM) [25] on the CM-5 [23] for our study. The CMAM message layer is analyzed and extended to expose the cost of communication functionality and network features. We consider several protocols ranging from a base single-packet message send and receive to more sophisticated protocols which provide services such as reliable multi-packet message delivery. These services incur software overhead because they require additional support in the messaging layer for buffer management, in-order delivery, and end-to-end fault tolerance.

Our study shows that this overhead accounts for 50–70% of the total cost and is a direct result of specific network features – arbitrary delivery order, finite buffering, and limited fault-tolerance. Since the CMAM messaging layer is already quite efficient, it is unlikely that this overhead can be reduced any further through improved software implementations. Because user communication requirements are relatively inflexible, we propose designing routing networks which change the constraints on the software layers, facilitating more efficient software (or even eliminating sections of them completely).

The contributions of this paper are:

- A framework which exposes the cost of the messaging layer.
- An assignment of these costs to user services and network hardware features.
- Identification of high level features which, if implemented in hardware, can significantly reduce the software cost of messaging.

The rest of the paper is organized as follows. Section 2 describes the problem context. In Section 3 we list the analysis framework, and justify our approach. Section 4 presents the details of the communication protocols and the corresponding breakdown of costs. Section 5 discusses and generalizes the results. A description of related work appears in Section 6, followed by our conclusions in Section 7.

2 Background

Communication in parallel machines requires both routing hardware and software messaging layers. The hardware provides the basic primitives, and the software layer orchestrates their use to provide application-level communication services. In this section, we discuss typical user communication services, current and likely future routing hardware features, and how a typical messaging layer bridges the two.

2.1 Communication Services

Application programs expect messaging layers to provide a minimal set of communication services. Most messaging layers provide the following services [5, 24, 12]:

- *Message Delivery*
- *Message Ordering*
- *Deadlock/Overflow Safety*
- *Reliable Delivery*

First, the most basic communication service is message delivery. Second, messages between a particular sender and receiver should be delivered in order of transmission. There is some debate about whether this is necessary, but it is both a commonly provided feature and a common programmer's assumption. Third, use of the network should not cause deadlock or the loss of data through some type of buffer overflow, relieving the programmer from explicitly managing the dynamic properties of the parallel system. Finally, messages should be delivered reliably. Most existing parallel machines settle for detecting errors and crashing. Given the expected mean-time-between-failures [15] of such machines, reliable communication is essential to their widespread use. These basic services ease programming concerns about the network for low-level explicit parallel programming (C or FORTRAN and message passing). They also support higher level approaches to programming parallel systems, allowing the compiler to ensure correct program execution without having to explicitly schedule and manage the network resources.

2.2 Routing Hardware Features

Though routing networks in commercial parallel machines (and in research literature) offer varied features, we focus on a few common features which are likely characteristics of future parallel machines and have significant impact on the software messaging layer:

- *Arbitrary Delivery Order*
- *Finite Buffering*
- *Fault-Detection but not Fault Tolerance*

Arbitrary delivery order means that the transmission order of messages between a particular source and destination are not preserved. This may arise because messages pass each other in the network (multipath routing (adaptivity) [9, 20], and virtual channels [7]), or because the network state may be swapped and resumed in a way that does not preserve transmission order (timesharing [10] and process migration). Finite buffering in networks and machines means that flow control is generally necessary for correct execution. Most networks provide deadlock-freedom guarantees based on the assumption that each output eventually absorbs any packets delivered to it, using software to ensure that all node always have enough space to absorb any messages sent to them. Most networks in parallel machines only provide error detection, but no error correction capabilities. This means that when a bad packet is detected, the entire computation must be aborted and perhaps the machine will also crash. In the long run, such behavior can incur significant time and computation loss (due to crashes), extra input/output for checkpointing, and poor availability. We believe that future machines will provide reliable message delivery.

2.3 Messaging Layers

Messaging layers bridge the gap between hardware features and communication services. Essentially, the messaging layer uses software protocols to provide any communication services not directly supported in hardware. Figure 1 shows the relationship between elements of a messaging layer, communication services, and network features. Each column in the box shows the messaging layer service needed to support the user requirement above it given the network features below. As can be seen, some of the services require extensive support.

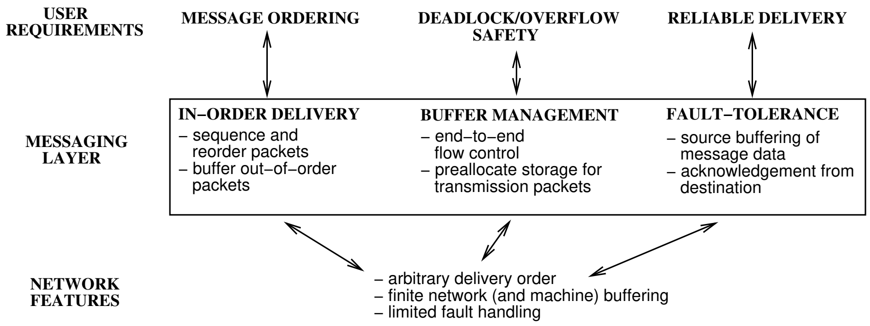

Fig. 1. Messaging layers bridge the gap between user requirements and network features.

Providing message ordering in a network that does not preserve transmission order means that the software must sequence outgoing packets and buffer packets arriving out of order. Ensuring deadlock/overflow safety in networks with finite network and node buffering means that the software must control buffer allocation and message transmission to avoid overcommitment of buffers. Generally this takes the form of preallocating space on the destination prior to transferring the message, ensuring that packets are introduced into the network only when they can be absorbed at the destination. Finally, providing reliable delivery in networks without error correction forces the messaging layer to keep copies of messages in transit. In addition, acknowledgements are also required to release these finite buffer resources.

The next section defines the specific framework used to analyze the costs of the messaging layer features described above.

3 Framework

Despite significant improvements in network interface design and advances in messaging implementations, software overhead of communication still dominates the hardware routing cost. To characterize the reasons for this overhead, we explore costs of communication services in a specific messaging layer, the *active messages* layer [25] on the Thinking Machines CM-5 (Connection Machine Active Messages or CMAM). The CM-5 provides user-level access to the network interface, which is essential for low-cost communication and likely to be a feature of future parallel machines. CMAM is widely recognized an extremely efficient messaging layer; in fact, several commercial vendors are extending their messaging layers to incorporate CMAM features. This efficiency is critical to the study, as our goal is to identify fundamental messaging costs, not that which result from a poor implementation. Further, the availability of the CMAM source code is a pragmatic concern, allowing us to accurately gauge and assign messaging cost, and also to assess the quality of the implementation.

The CM-5 Network To explain how CMAM works, we first describe the CM-5 network interface (NI) and communication mechanisms. The CM-5 NI provides a memory-mapped interface (control registers and network FIFO's) on the processor-memory bus (see Figure 2). A packet is injected into the network by storing the destination node number and data arguments to the NI send buffer. Packets are extracted using LOADs from the receive buffer, while the NI status is queried by loading the control registers.[1]

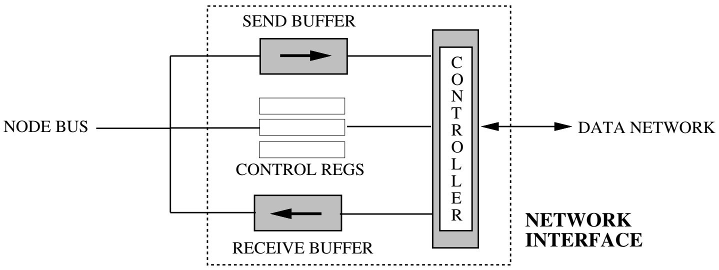

Fig. 2. The CM-5 network interface.

The CM-5 network has a number of features which increase the messaging layer complexity. First, out-of-order delivery requires the messaging layer to sequence and reorder packets. Second, the network interfaces and network have finite buffering, so software buffer management is required. Third, the CM-5 network provides error detection at the packet level, but no error correction, requiring a software protocol to ensure reliable delivery. And finally, the CM-5 network hardware only supports packets with five 32-bit words, so a typical message is broken into many packets, further increasing the software overhead.

CMAM: A Simple Messaging Layer The basic communication primitive provided by CMAM is an active message: a message with an associated small amount of computation (in the form of a handler) at the receiving end. The current CMAM implementation polls the network to accept messages.[2] We use two CMAM interfaces in our study. The first interface provides an active message which carries up to four words of user data and is implemented by the CMAM_4 function at the source, and CMAM_request_poll, CMAM_handle_left, and CMAM_got_left functions at the destination. These functions provide a minimal layer over the raw hardware for injecting and extracting packets from the network. The second interface supports bulk memory-to-memory transfers, and is implemented using CMAM_xfer function which splits up the transfer into a sequence of hardware packets at the source, and CMAM_handle_left_xfer function which reassembles the packets at the destination.

[1] While this is not the most efficient type of network interface [13, 8, 4], it has the significant virtue that no changes to the processor are required. Many researchers believe that this type of interface is basically representative of future network interfaces.

[2] The CM-5 NI also supports an interrupt-driven interface for reception; however, the cost is very high for interrupts in the SPARC processor.

4 Analysis of messaging layer

Using CMAM, we examine the costs for implementing typical communication protocols, at each stage relating the measured costs to the network features. To estimate the costs of different portions of the CMAM layer, we use dynamic instruction counts of the CMAM assembly code (some of CMAM is coded in assembly and the rest was generated from the original C code using gcc -O2). Execution paths which minimize the instruction count are chosen, so our measurements are an optimistic estimate of the actual software overhead.

Although actual execution time might be a better metric for characterizing communication cost, we use instruction counts consistently throughout the paper. There are two reasons for this: First, obtaining instruction counts is much simpler than counting actual cycles. This is because the cycle counts depend on details such as write buffers, bus synchronization, memory refresh, and other concerns. Second, the instruction counts represent a more useful and portable characterization of the messaging costs. While the times for each instruction may vary depending on the implementation, it appears likely that the instruction counts will be substantially similar across machines with memory-mapped network interfaces.

We consider the implementation of three communication protocols:

1. **Single-packet delivery** sends and receives a single-packet message.
2. **Finite-sequence multi-packet delivery** sends and receives a fixed-size user message which consists of a several hardware packets.
3. **Indefinite-sequence multi-packet delivery** sends and receives an indefinite sequence of packets, corresponding to an ordered stream of communication.

Single-packet delivery is the cheapest kind of communication possible in the CM-5 – a four word datagram packet. The multi-packet protocols have a base cost based on single packet deliveries but incur additional overhead to support user communication requirements. Finite-sequence multi-packet delivery incurs the costs of buffer management and in-order delivery to support larger messages, preallocating buffers for all injected packets and sequencing and reordering the received packets. Indefinite-sequence multi-packet delivery sees the additional cost of in-order delivery for buffering out-of-order packets at the destination. Both multi-packet protocols also have additional overhead for ensuring reliable transmission.

We now examine these protocols in detail: for each protocol, we first give details of the implementation, and then a breakdown of the measured costs.

4.1 Single-packet delivery

This protocol consists of sending and receiving a message consisting of a single packet. The packet is sent using the CMAM_4 function, and received using a combination of the CMAM_request_poll, CMAM_handle_left, and CMAM_got_left functions. The receiver functions, respectively, check for any outstanding packets, receive waiting packets, and invoke the user handlers for received packets.

Table 1 presents a breakdown of the source and destination costs. Source costs include setting up the network interface (NI), writing user data, and polling a NI

Description	*Source*	*Destination*
Call/Return	3	10
NI setup	5	–
Writing to NI	2	–
Reading from NI	–	3
Checking NI status	7	12
Control flow	3	2
	20	27

Table 1. Instruction counts for single-packet delivery.

status register which confirms the send as well as tests for presence of any incoming packets. The destination overhead includes polling the NI to check for any waiting packets, extracting the packet data, vectoring on the hardware message tag, and then invoking the user handler. Thus, single-packet transfer in the CMAM layer is extremely efficient, costing only 47 instructions, of which 34 instructions are dedicated to accessing the network interface. This number is essentially the minimum required to interface with the hardware on a CM-5. However, the single-packet protocol does not meet any of the requirements for communication services – the packets are not ordered, nor are they deadlock/overflow safe, nor are they delivered reliably. Further, communication can only be done in units of four data words.

4.2 Finite-sequence multi-packet delivery

This protocol supports messages of arbitrary size and transfers data from a buffer in the source memory to a buffer at the destination. This protocol is implemented using the CMAM_xfer_N and CMAM_handle_left_xfer functions, and consists of six steps (see Figure 3). The sender first transmits a buffer allocation request to the receiver (Step 1), which allocates a buffer and associates it with a communication segment (Step 2). On receiving the response (Step 3), the sender initiates a sequence of single-packet transfers (Step 4). The receiver node stores the packet data into the allocated segment. On completion of the transfer, the receiver frees the communication segment (Step 5), and sends back an acknowledgement packet (Step 6).

The protocol cost has four components: base single-packet transfers (Step 4), buffer management (Steps 1,2,3 and 5), in-order delivery, and fault-tolerance (Step 6). Single-packet transfers move data from source to destination, contributing a single-packet delivery and reception cost for each transfer. The base cost also reflects the additional LOADs/STOREs required for the memory-to-memory transfer.

Buffer management provides deadlock and overflow safety by preallocating buffer space for messages on their destination nodes. The preallocation cost involves two single-packet deliveries (request and response), and the cost of associating a segment number with the target buffer. The buffer also needs to be deallocated at the end of the transfer (to permit reuse).

In-order delivery ensures that the data gets written at the correct position in the destination's buffer. Its cost arises from the following protocol: each packet carries an offset into the target buffer where its data should be stored. The destination

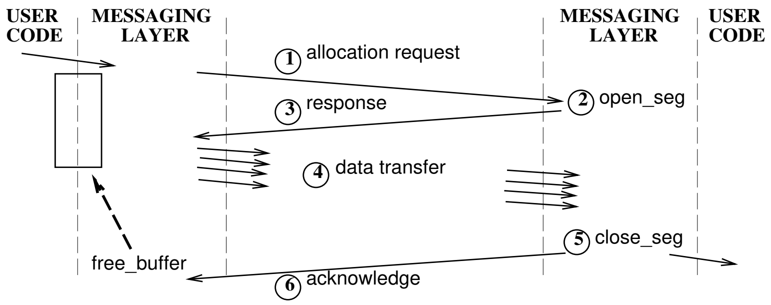

Fig. 3. Protocol for finite-sequence multi-packet delivery.

extracts this offset and updates the communication segment count associated with the transfer. The source overheads include incrementing and storing the offset for each packet, while the receiver overheads include a LOAD to extract the offset and operations to decrement the count. In-order delivery support masks the arbitrary packet delivery order inherent in the CM-5 network.

Fault tolerance ensures that a copy of the data is maintained at the source pending acknowledgement of successful reception. Its costs are that of a single-packet delivery acknowledging successful completion of transfer (source buffering has no cost in our accounting since it can be implemented by not releasing the original message buffer until the acknowledgement is received).

These four components of finite-sequence multi-packet delivery costs are shown on the left in Tables 2 and 3 for two message sizes: small (16 words) and large (1024 words). To obtain the best possible execution path, we assume there is no other communication going on at the source and destination nodes. We observe that buffer management contributes 50% of the total transfer costs for small messages because of the round-trip handshake, but has negligible impact on the communication cost for large messages. Despite this, additional messaging layer costs still account for ~10% of the costs in the latter case. In-order delivery does not contribute as much to the total cost since the requirements on ordering are relaxed: instead of specifying an exact order of arrival, the only user requirement is that the data be placed in the target buffer in order. Knowing the total count of expected packets allows this functionality to be provided cheaply. As we will see in the next section, sequencing can be quite expensive if this information is not available. Fault-tolerance accounts for ~10% of the total cost for the small transfer, but has negligible impact on the costs of the large transfer.

4.3 Indefinite-sequence multi-packet delivery

This protocol supports the sending and receiving of an indefinite length sequence of hardware packets between a pair of nodes. These packets are all part of a larger logical user communication. Such a communication pattern is normally associated with static channels between a pair of user processes (sockets) and is characterized by an indefinite amount of communication through the channels.

Figure 4 shows the protocol steps. The sender first buffers the user message

Message size = 16 words

Finite-sequence multi-packet delivery

Feature	Source	Destination	Total
Base Cost	91	90	181
Buffer Mgmt.	47	101	148
In-order Del.	8	13	21
Fault-toler.	27	20	47
Total	173	224	397

Indefinite-sequence multi-packet delivery

Feature	Source	Destination	Total
Base Cost	80	69	149
Buffer Mgmt.	–	–	–
In-order Del.	20	116	136
Fault-toler.	116	80	196
Total	216	265	481

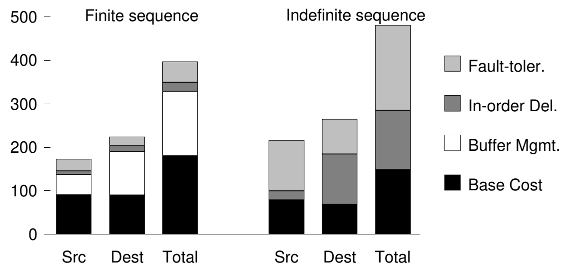

Table 2. Multi-packet delivery: packet size = 4 words, message size = 16 words.

Message size = 1024 words

Finite-sequence multi-packet delivery

Feature	Source	Destination	Total
Base Cost	5635	4626	10261
Buffer Mgmt.	47	101	148
In-order Del.	512	769	1281
Fault-toler.	27	20	47
Total	6221	5516	11737

Indefinite-sequence multi-packet delivery

Feature	Source	Destination	Total
Base Cost	5120	3597	8717
Buffer Mgmt.	–	–	–
In-order Del.	1280	7424	8704
Fault-toler.	7424	5120	12544
Total	13824	16141	29965

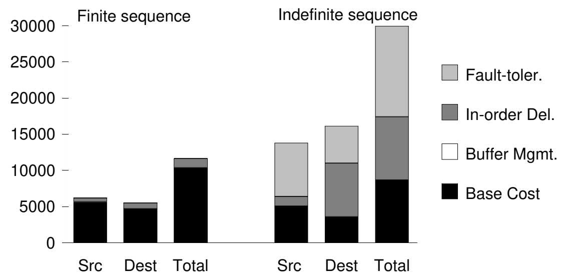

Table 3. Multi-packet delivery: packet size = 4 words, message size = 1024 words.

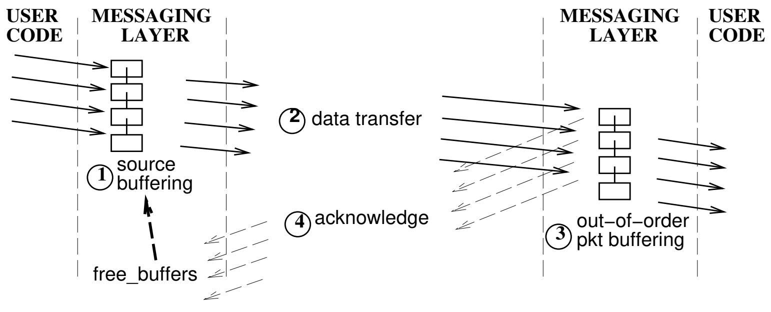

Fig. 4. Indefinite-sequence multi-packet protocol.

(Step 1) (to support retransmission if required), and then sends it using single-packet transfers (Step 2). The receiver node buffers all out-of-order packets (Step 3), initiating the appropriate user handler for each packet arriving in its transmission order. Since the entire transmission may be very large, each packet has its own acknowledgement (Step 4), which allows source storage to be released.

As in Section 4.2, this protocol also has four components contributing to the total cost: base single-packet transfers, buffer management, in-order delivery, and fault-tolerance. The base cost consists of single-packet delivery and reception costs for each transfer.

Buffer management may be required for three reasons: to buffer user messages at the source, to handle out-of-order packets at the receiver, and to store user data into a receiver buffer. We have chosen to account for the source buffering costs as part of the fault-tolerance overheads, and the out-of-order packet buffering as part of the in-order delivery costs. Further, since the user view of register-to-register communication eliminates the need for a separate buffer, buffer management has negligible cost in this protocol[3]

In-order delivery incurs a transmission overhead of sequence numbers and requires the buffering of out-of-order packets at the destination since there is no space allocated for them elsewhere. Note that unlike the finite-sequence multi-packet protocol, one cannot exploit information about the number of expected packets to reduce sequencing costs.

Fault-tolerance costs arise from source buffering and acknowledgement messages. Source buffering contributes additional STOREs to the cost, and supports retransmission in the presence of faults. Each acknowledgement message incurs the cost of a single-packet delivery. For larger (and more predictable) packet sequences, this per-packet cost can be reduced by employing group acknowledgements (at the cost of reserving source buffers for a longer amount of time).

These components of indefinite-sequence multi-packet delivery costs are shown on the right in Tables 3 for the two message sizes of 16 and 1024 words. Message sizes correspond to the total data volume transmitted. To measure in-order delivery costs, we assume that half of the packets arrive out of order. We observe from the

[3] Some systems may include buffer management into this protocol to provide overflow safety. This only introduces additional buffer management overhead, but does not affect the cost of the other features.

costs that the in-order delivery functionality accounts for > 50% of the end-to-end costs, and this fraction is independent of the total volume of data transmitted.

The breakdown of costs for the two multi-packet delivery protocols shows that a large fraction of the end-to-end cost is attributable to communication services – in-order delivery, deadlock and overflow safety, and fault-tolerance – and accounts for 50–70% of the total cost in all situations except large finite-sequence multi-packet transfers. Conversely, this functionality can also be viewed as the cost of specific network features – arbitrary delivery order, finite network and node buffering, and limited fault-handling capabilities. This suggests that the end-to-end communication cost can be significantly improved if the underlying network provides more sophisticated functionality, replacing its implementation in software.

5 Discussion

No single study can address all of the issues. In this study, we have chosen to focus on basic delivery, ordering, buffer management, and fault tolerance, while putting aside issues of protection and scheduling. Clearly, any solution which reduces messaging overhead to the level of tens of instructions must deal with these issues.

Studying a particular network, network interface, and messaging layer limits the applicability of our conclusions to a range of similar systems. These concerns can only be mitigated by studying a system that is as representative as possible so the conclusions are broadly applicable. We believe that CMAM on the CM-5 is representative of many future messaging systems. However, in the following paragraphs we consider the effect on our results of modest changes in machine architecture.

Larger packet sizes CMAM on the first version of the CM-5 network interface supports four data words per packet; however, other parallel machines and even the second version of the CM-5 network interface support larger packet sizes. Figure 5 shows the variation in messaging overhead for a message size of 1024 words as the packet size is varied from 4–128 words (assuming that the rest of the network mechanisms remain the same). It is clear that for indefinite-sequence multi-packet delivery, the messaging overhead constitutes a significant fraction over the entire range. For finite-sequence multi-packet deliveries, this overhead is lower, but still significant, accounting for 9–11% of the total cost.

Direct memory access (DMA) hardware One notable shortcoming of the CM-5 network hardware is the lack of any DMA capability. While DMA can reduce the cost of moving large amounts of data, it is unlikely that it would give much benefit for the packet sizes we considered (4–128 words). Further, as with other improvements to the network interface, it would affect only the base cost portions of our instruction counts. If the base cost is reduced, that increases the importance of the costs in the rest of the messaging layer.

Implications for Network Design One important contribution of this study is to concretely establish the cost and benefit of a variety of network features. In many

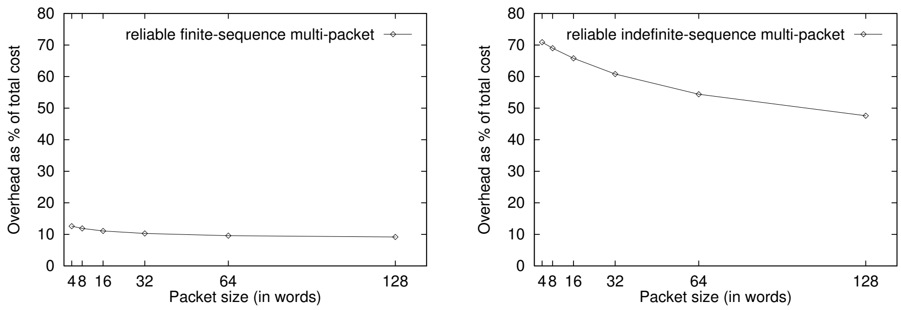

Fig. 5. Messaging layer overhead versus packet size for 1024 words of communication.

cases, there is a tension between optimizing routing performance, and improving end-to-end communication performance. As we have seen in this study, some features which produce improved routing performance may incur unacceptable software costs. For example, a number of designs have proposed out-of-order delivery in order to improve network routing performance (randomization [18, 1] or adaptive routing [20, 7, 9]). Our results show clearly that a significant cost can be incurred for packet sequencing and reordering, so the benefits of out-of-order delivery for the network must be weighed against the software costs of such behavior. Because the software overhead is generally much larger than the hardware routing time, in many cases, the overheads of such features will outweigh their benefits.

Another perspective is that our results point out where "high level" network features can be of most benefit. Such features can eliminate the need for parts of the messaging layer, reducing communication cost. Specifically, if a network could provide in-order delivery, end-to-end flow control, and reliable delivery, it would obviate much of the need for a messaging layer, reducing the messaging cost to the basic data movement cost. Researchers in the routing community are beginning to address such concerns; several recent projects (e.g., Compressionless Routing [17]) provide all three of these features, albeit at some cost.

Communication Cost versus Latency In this study, we have focused on instruction counts as the primary measure of communication cost (performance). Latency is a reasonable alternative performance metric, but is hard to measure in a portable fashion because it depends on a host of low-level system detail as well as software policies such as scheduling. For cases where software overhead dominates, instruction counts are indicative of communication latency.

6 Related Work

A great deal of attention has been focused on the design of interconnection networks, network interfaces, and even fast messaging layers. The main distinction is that our study seeks to implement high level software communication primitives without compromising on performance or functionality.

Research on interconnection networks has focused primarily on optimizing for bandwidth and latency performance metrics [21, 6, 22, 10]. While these are certainly important attributes, much less energy has gone into exploring what impact advanced network features (adaptive routing, virtual channels) have on network interface complexity and software overhead. Our work addresses some of these issues.

Research on network interfaces has focused primarily on reducing message injection (and reception) overhead [13, 8, 19, 4] or offloading the communication onto a coprocessor [14, 16, 3]. Such efforts are complementary to our goal of software protocol overhead reduction. Improvements in network interface can reduce the basic communication cost in our studies. While reducing the basic cost is important, as can be seen from our studies, reducing the software protocol overhead is equally important. Further, reductions in the basic cost will increase the importance of reducing software protocol overhead.

Researchers have explored a variety of approaches for reducing the cost of messaging layers. While substantial progress has been made, much of it has been made at the cost of reducing functionality. For example, techniques for speeding interprocess communication [2, 11, 26] have resorted to lower level (and more risky) communication primitives to achieve high performance. In parallel systems, a number of reduced messaging layers have also been developed. In fact, active messages, the basis for our study, is one such reduced layer. The lowest level primitive – single packet send – is widely used, but is unsafe because no flow control is performed. While reduced functionality layers may be a necessary expedient, a better long term solution would involve full functionality communication primitives and high performance.

7 Conclusions

We have described a study to measure the end-to-end costs of communication by analyzing the costs incurred by a prototypical messaging layer, specifically the CMAM layer running on the CM-5. Since messaging layers must bridge the gap between the hardware and user communication requirements, the costs in a messaging layer often reflect the costs of particular network features. Our study provides a piecewise breakdown of the communication costs and an assignment of that cost the various network features and communication services that incur it.

Our results show that 50–70% of end-to-end messaging cost is attributable to arbitrary order of packet delivery, finite buffering, and limited fault-handling capabilities. And although the CMAM layer is extremely efficient – virtually all of the remaining time is spent copying data to and from the network interfaces – the cost of transmitting a 16-word message is between 285 and 481 instructions with the network features accounting for a major share. While improvements to the CM-5's network interface are possible, paradoxically, such improvements will only exacerbate the situation, causing these features to constitute an even larger percentage of the cost.

There is often a tradeoff between optimizing routing performance and reducing software overhead. As we have seen, some network features may well incur software overheads that outweigh the features' benefits. Because the software overhead is often a dominant contributor, network designers must also consider how their decisions affect the software messaging layers. One possible direction for this work is to broaden the study to a number of parallel systems. Another avenue for future work is to design hardware which provides functionality that eliminates the need for or simplifies the messaging layer.

Acknowledgements

The authors would like to thank Jae-Hoon Kim and John Plevyak for participating in several discussions about the messaging protocols described in the paper.

The research described in this paper was supported in part by National Science Foundation grant CCR-9209336, Office of Naval Research grants N00014-92-J-1961 and N00014-93-1-1086, and National Aeronautics and Space Administration grant NAG 1-613.

References

1. G. Alverson, R. Alverson, D. Callahan, B. Koblenz, A. Porterfield, and B. Smith. Exploiting heterogeneous parallelism on a multithreaded multiprocessor. In *Proceedings of the 6th ACM Interational Conference on Supercomputing*, 1992.
2. B. N. Bershad, T. E. Anderson, E. D. Lazowska, and H. M. Levy. User-level interprocess communication for shared memory multiprocessors. *ACM Transactions on Computer Systems*, 9(2):175–198, May 1991.
3. M. A. Blumrich, Kai Li, Richard Alpert, Cezary Dubnicki, and Edward W. Felten. Virtual memory mapped network interface for the shrimp multicomputer. In *Proceeding of the International Symposium on Computer Architecture*, April 1994.
4. S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H. T. Kung, M. Lam, B. Moore, C. Peterson, J. Pieper, L. Rankin, P. S. Tseng, J. Sutton, J. Urbanski, and J. Webb. iWarp: An integrated solution to high-speed parallel computing. In *Proceedings of Supercomputing '88*, pages 330–341. IEEE Press, 1988. Orlando, Florida.
5. V. Cerf and R. Kahn. A protocol for packet network interconnection. *IEEE Transactions on Communications*, 1974.
6. W. Dally and C. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5), 1987.
7. W. J. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks usin g virtual channels. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):466–74, April 1993.
8. W. J. Dally, A. Chien, S. Fiske, W. Horwat, J. Keen, M. Larivee, R. Lethin, P. Nuth, S. Wills, P. Carrick, and G. Fyler. The J-Machine: A fine-grain concurrent computer. In *Information Processing 89, Proceedings of the IFIP Congress*, pages 1147–1153, August 1989.
9. J. Duato. On the design of deadlock-free adaptive routing algorithms for multicomputers: design methodologies. In *Proceedings of Parallel Architectures and Languages Europe*, 1991.
10. C. Leiserson et al. The network architecture of the Connection Machine CM-5. In *Proceedings of the Symposium on Parallel Algorithms and Architectures*, 1992.
11. Robert Fitzgerald and Richard F. Rashid. The integration of virtual memory management and interprocess communication in accent. *ACM Transactions on Computer Systems*, 4(2):147–77, May 1986.
12. G. Geist and V. Sunderam. The pvm system: Supercomputer level concurrent computation on a heterogeneous network of workstations. In *Proceedings of the Sixth Distributed Memory Computers Conference*, pages 258–61, 1991.

13. D. S. Henry and C. F. Joerg. A tightly-coupled processor-network interface. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages an Operating Systems*, pages 111–122, 1992.
14. M. Homewood and M. McLaren. Meiko CS-2 interconnect Elan – Elite design. In *Proceedings of the IEEE Hot Interconnects Symposium.* IEEE TCMM, August 1993.
15. B. Horst. Massively-parallel systems you can trust. In *Proceedings of COMPCON Spring '94.* IEEE Computer Society, IEEE Press, February 1994.
16. Intel Corporation. *Paragon XP/S Product Overview*, 1991.
17. J. Kim, Z. Liu, and A. Chien. Compressionless routing: A framework for adaptive and fault-tolerant routing. In *Proceedings of the International Symposium on Computer Architecture*, April 1994.
18. S. Konstantinidou and L. Snyder. Chaos router: Architecture and performance. In *Proceedings of the International Symposium on Computer Architecture*, pages 212–21, 1991.
19. J. Kubiatowicz and A. Agarwal. Anatomy of a message send in alewife. In *Proceedings of the International Conference on Supercomputing*, 1993.
20. L. Ni and C. Glass. The turn model for adaptive routing. In *Proceedings of the International Symposium on Computer Architecture*, 1992.
21. J. A. Patel. Procesor-memory interconnections for multiprocessors. In *Proceedings of the 6th Annual Symposium on Computer Architecture*, pages 168–177. IEEE Computer Society, IEEE Press, April 1979.
22. C. Seitz and W. Su. A family of routing and communication chips based on the Mosaic. In *Proceedings of the University of Washington Symposium on Integrated Systems*, 1993.
23. Thinking Machines Corporation, Cambridge, Massachusetts. *Connection Machine CM-5, Technical Summary*, November 1992.
24. Thinking Machines Corporation, Cambridge, Massachusetts. *CMMD Reference Manual, V3.0*, May 1993.
25. T. von Eicken, D. Culler, S. Goldstein, and K. Schauser. Active Messages: a mechanism for integrated communication and computation. In *Proceedings of the International Symposium on Computer Architecture*, 1992.
26. M. Young, A. Tevanian, R. Rashid, D. Golub, J. Eppinger, J. Chew, W. Bolosky, D. Black, and R. Baron. The duality of memory and communication in the implementation of a multiprocessor operating system. In *Proceedings of the Eleventh Symposium on Operating Systems Principles*, pages 63–76. ACM, 1987.

This article was processed using the LaTeX macro package with LLNCS style