In Proceedings of ASPLOS-VI, San Jose, California, October 5-7, 1994.

Software Overhead in Messaging Layers: Where Does the Time Go?

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 improvements in network interfaces and software messaging layers, software communication overhead still dominates the hardware routing cost in most systems. In this study, we identify the sources of this overhead by analyzing software costs of typical communication protocols built atop the active messages layer on the CM-5. We show that up to 50–70% of the software messaging costs are a direct consequence of the gap between specific network features such as arbitrary delivery order, finite buffering, and limited fault-handling, and the user communication requirements of in-order delivery, end-to-end flow control, and reliable transmission. However, virtually all of these costs can be eliminated if routing networks provide higher-level services such as in-order delivery, end-to-end flow control, and packet-level fault-tolerance. We conclude that significant cost reductions require changing the constraints on messaging layers: we propose designing networks and network interfaces which simplify or replace software for implementing user communication requirements.

1 Introduction

In highly parallel machines, a collection of computing nodes work in concert to solve large application problems, coordinating their efforts by sending and receiving messages through the communication network. Thus, the achievable performance of such machines critically depends on the end-to-end cost of communication mechanisms. The main cost contributors are 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 [26] and improved network interfaces [24, 13] have significantly reduced the software cost of messaging. However, software communication overhead continues to dominate the hardware routing cost. Our study focuses on understanding the reasons for software overhead by precisely characterizing the costs in a prototypical messaging layer. Messaging layers on parallel machines bridge the gap between raw hardware functionality and the higher-level user communication requirements. Thus, the cost characterization can be viewed either as the cost of messaging layer services, or alternately, as the cost of particular network hardware features which necessitate additional software to provide critical application services.

This study has two parts. In the first part, we analyze the costs of communication functionality and network features in typical communication protocols built atop a minimal and efficient messaging layer, the CM-5 [24] *active messages* layer (CMAM) [26]. We first consider a base protocol (single-packet delivery), and then examine more sophisticated protocols providing reliable multi-packet message delivery. The latter protocols incur software overhead because they require additional support for buffer management, in-order delivery, and end-to-end fault tolerance. Our study shows that this overhead accounts for 50 – 70% of the software communication cost and is a direct result of specific CM-5 network features – arbitrary delivery order, finite buffering, and limited fault-tolerance.

Given that CMAM is already quite efficient, we are left with two alternatives for reducing this overhead: either lower the level of user communication services, or raise the level of services provided by the network. In the second part of this study, we examine what overheads might be recovered with routing networks which exploit low-level hardware structure to provide higher-level services. Using a network design based on *Compressionless Routing* [16] which provides in-order delivery, end-to-end flow control, and packet-level fault-tolerance, we show that the software overheads of buffer-management, in-order delivery and end-to-end fault-tolerance can be completely eliminated. Consequently, we propose designing networks which simplify the messaging software (or replace some portions). This means that applications can use high-level communication services without sacrificing efficiency.

The specific contributions of this paper include:

- A detailed analysis of an efficient messaging layer which identifies the key cost components and attributes them to specific user communication services.
- Correlation of the cost analysis with network hardware features, pointing out the software cost of several commonly discussed routing features. Network designers

must weigh the software cost of these features against their hardware performance benefits.

- A messaging layer implementation which demonstrates that if a network provides high level services (in-order delivery, end-to-end flow control, and reliable packet delivery), essentially all but the data movement cost can be removed from the messaging layer. Several networks with such features have been proposed [22, 16].

The remainder of the paper is organized as follows. Section 2 describes the problem context. In Section 3, we analyze the costs incurred by CMAM-based implementations of typical communication protocols. Section 4 explores the advantages of higher level network features on messaging layer costs. We discuss and generalize our results in Section 5. A description of related work appears in Section 6, followed by our conclusions in Section 7.

2 Background

Communication in parallel machines [24, 15, 13] requires both routing hardware which supplies the basic primitives, and a software messaging layer which orchestrates their use to provide application-level communication services. In this section, we discuss typical application-level services, current and likely future routing hardware features, and how a typical messaging layer bridges the two.

2.1 Communication services

Application programs typically expect a basic set of communication services from messaging layers. Most messaging layers provide the following services [3, 25, 11, 10]:

1. *Message Delivery*
2. *Message Ordering*
3. *Deadlock/Overflow Safety*
4. *Reliable Delivery*

First, the most basic service is data movement from the sender to the receiver. Second, messages between a particular sender and receiver should be delivered in order of transmission. Although not strictly necessary, this is both a commonly provided feature and a common programmer's assumption. Third, use of the network should not cause deadlock or data loss through buffer overflow. This is essential; without it programmers would be forced to reason about scheduling and resource usage in detail. Finally, messages should be delivered reliably. Most existing parallel machines settle for detecting errors and crashing. However, given their exhibited mean-time-between-failures [14], the need for reliable communication is evident. These basic services ease network programming concerns for low-level explicitly parallel programming (C or FORTRAN and message passing). They also support higher level approaches to programming parallel systems, freeing the compiler from having to explicitly schedule and manage the network resources for correct program execution.

2.2 Routing hardware features

Though routing networks in commercial machines [24, 15] (and in literature) offer a variety of features, we focus on those which are likely characteristics of future systems, and have significant impact on the software messaging layer:

1. *Arbitrary Delivery Order*
2. *Finite Buffering*
3. *Fault-detection but not Fault-tolerance*

Arbitrary delivery order means that the transmission order of messages between a particular source and destination is not preserved. This may arise when messages pass each other in the network (as with multipath routing (adaptivity) [7, 20] and virtual channels [4]), or when the network state is swapped and resumed in a way that does not preserve delivery order (as with timesharing [8] and process migration). Finite buffering in machines means that flow control is generally necessary for correct execution. Most networks provide deadlock-freedom guarantees on the assumption that each output eventually extracts any packets delivered to it, relying on software to ensure that all nodes always have enough space to absorb incoming packets. Most networks provide only 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, extra input/output (due to checkpointing), and poor availability.

2.3 Software 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 messaging layer elements, user services, and network features. Each column 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.

Providing in-order delivery in a network which does not preserve transmission order means that the software must sequence outgoing packets and buffer incoming packets arriving out of order. Ensuring deadlock/overflow safety in networks with finite network and node buffering means that a messaging layer must avoid over-commitment of buffers. Generally this involves preallocating space on the destination, 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 analyzes a specific messaging layer to characterize the costs of the features described above.

3 Analysis of messaging layer costs

We explore costs of communication services in a specific messaging layer, the CM-5 *active messages* layer [26]

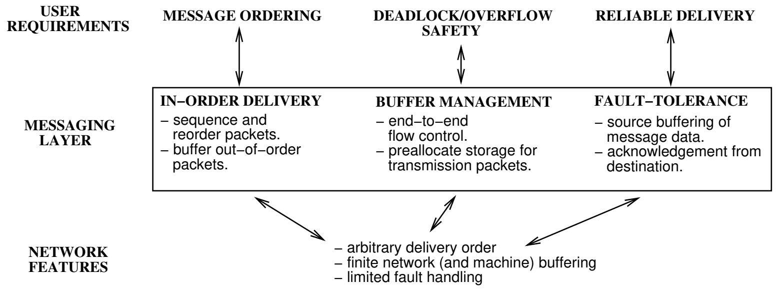


Figure 1: *Messaging layers bridge the gap between user requirements and network features.*

(CMAM). User-level access to the CM-5 network interface (NI) is essential for low-cost communication and is a likely feature of future parallel machines. CMAM is widely recognized as efficient; in fact, several commercial vendors are extending their messaging layers to incorporate CMAM features. CMAM's efficiency is critical as our goal is to identify fundamental messaging costs, not those which result from poor implementation. In addition, the availability of the CMAM source code is a pragmatic concern, allowing us to accurately gauge and assign messaging cost.

We first give the relevant background by describing the CM-5 NI, the CM-5 network, and CMAM interfaces. We then analyze software costs incurred by typical communication patterns implemented using the CMAM layer.

3.1 The CM-5 and CMAM

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 NI receive buffer, while the NI status is queried by loading the control registers.[1]

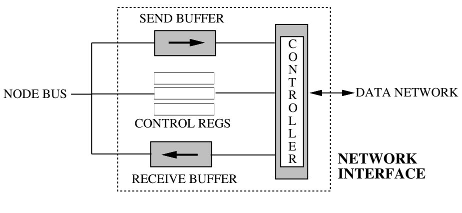


Figure 2: *The CM-5 network interface.*

The CM-5 network [8] 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 nodes, NI, and the network all 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 provides a simple messaging layer on the CM-5. The basic primitive 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 implementation polls the network to accept messages.[2] We use two CMAM interfaces in our study. The first 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. The second interface supports bulk memory-to-memory transfers and is implemented using the CMAM_xfer function which splits up the transfer into a sequence of hardware packets at the source, and the CMAM_handle_left_xfer function which reassembles the packets at the destination.

3.2 Communication costs of typical protocols

Using CMAM, we examine the costs for implementing typical protocols, at each stage relating the measured costs to the network features. Costs were measured using 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 a conservative estimate.

Although actual execution time might be a better metric for characterizing communication cost, we use instruction counts for three reasons. First, obtaining accurate cycle counts is not tractable as they depend on details such as write buffers, bus synchronization, memory refresh, and even the user application. Second, instruction counts represent a portable and perhaps more useful 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 the same across machines with memory-mapped network interfaces. Finally, instruction counts lend themselves to use with simple models which allow determination of the messaging overhead for the machines and applications of interest.

[1] While this is not the most efficient type of network interface [12, 6], it requires no changes to the processor. Many researchers believe that this type of interface is representative of future network interfaces.

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

A detailed breakdown of the instruction counts into several subcategories for the protocols described in this section appears in Appendix A. In the rest of this paper, we confine ourselves to a simple model where all instructions are assumed to have unit cost.

We consider the implementation of three protocols:

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

Single-packet delivery is the cheapest communication possible in CMAM – 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 buffer management and in-order delivery costs to support larger messages. Indefinite sequence, multi-packet delivery sees additional in-order delivery costs. 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.

Single-packet delivery This protocol consists of sending and receiving a single-packet message. 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 outstanding packets, receive waiting packets, and invoke user handlers for received packets.

Description	*Source*	*Destination*
Call/Return	3	10
NI setup	5	–
Write to NI	2	–
Read from NI	–	3
Check NI status	7	12
Control flow	3	2
	20	27

Table 1: *Instruction counts for single-packet delivery.*

Table 1 presents a breakdown of the source and destination costs. Source costs include NI setup, writing user data to the NI send buffer, and polling a status register which confirms the send as well as tests for presence of any incoming packets. The destination costs include 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 costs 47 instructions, of which 34 instructions are dedicated to accessing the NI. This number is essentially the minimum required to interface with the CM-5 hardware. However, this protocol does not meet any of the requirements for communication services – 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.

Finite sequence, multi-packet delivery This protocol supports arbitrary size messages and reliably transfers data from a source memory buffer to a destination memory buffer. The protocol is implemented using CMAM_xfer_N and CMAM_handle_left_xfer functions and consists of six steps (see Figure 3). The sender sends an allocation request to the receiver (Step 1), which allocates a communication segment (Step 2) and replies to the sender (Step 3). The sender then initiates a sequence of single-packet transfers (Step 4), whose data are stored by the receiver into the allocated segment. On completion of the transfer, the receiver frees up the communication segment (Step 5), and sends back an acknowledgement packet (Step 6).

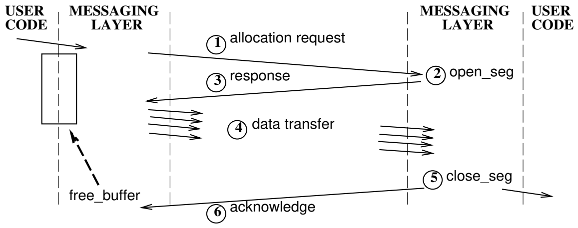


Figure 3: *Finite sequence, multi-packet protocol.*

The protocol cost has four parts: 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 contribute a single-packet send and receive cost per transfer. The base cost also includes the additional LOAD/STOREs to move data up/down the memory hierarchy.

Buffer management costs include the preallocation and deallocation of the destination buffer. The preallocation cost involves two single-packet deliveries (request and response), and the cost of associating a segment number with the target buffer. Deallocation disassociates the segment number from the target buffer.

In-order delivery ensures that data is written at the correct position in the destination buffer. Its cost arises from the following protocol: each packet carries an offset into the target buffer where it should be stored, eliminating the need for sequence numbers. The destination 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.

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 the transfer.

These four components of finite sequence, multi-packet delivery costs are shown in Table 2 for two message sizes: small (16 words) and large (1024 words). To obtain the most favorable 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

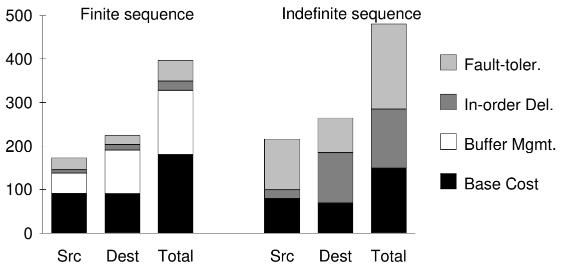
Message size = 16 words

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



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

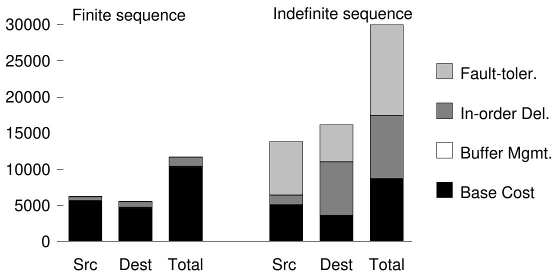


Table 2: *Multi-packet delivery costs for 16- and 1024-word messages: packet size = 4 words.*

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.

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

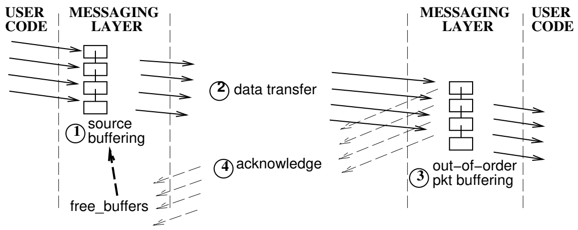


Figure 4: *Indefinite sequence, multi-packet protocol.*

As in the finite sequence protocol, this protocol also has four components contributing to the total cost: base single-packet transfers (Step 2), buffer management, in-order delivery (Step 3), and fault-tolerance (Steps 1 and 4). The base cost consists of single-packet delivery and reception costs for each transfer.

In this protocol, buffer management refers to one of three kinds of buffers: those used to source-buffer messages, those used to handle out-of-order packets at the receiver, and those required to store user data into a receiver buffer. We have chosen to account for source-buffering 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 receiver buffer, buffer management has negligible cost in this protocol.[3]

In-order delivery incurs a source 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. 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) messages, this per-packet cost can be reduced by employing group acknowledgements (at the cost of reserving source buffers for a longer period of time).

These components of indefinite sequence, multi-packet delivery costs are shown in Table 2 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 the packets arrive out of order. We observe from the costs that the in-order delivery and fault-tolerance functionality accounts for $\sim$70% of the end-to-end costs, and this fraction is independent of the total volume of data transmitted. As can be seen, the overhead remains significant ($\sim$40–50%) even if group acknowledgements are employed.

3.3 Summary

The breakdown of costs for the two multi-packet delivery protocols shows that a large fraction of the end-to-end software communication 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. Another perspective on these results is to view these overheads as costs of specific network features – arbitrary delivery order, finite network and node buffering, and limited fault-handling capabilities. Since the CMAM layer is already quite efficient, these costs are unlikely to be eliminated through improved software implementations.

Thus, there are two alternatives for reducing these costs: either lower the level of user communication services, or raise the level of services provided by the network. The former alternative is, in general, undesirable because it requires parallel software to build messaging services and manage their cost as appropriate. In the next section, we consider the costs of the same multi-packet protocols using a messaging layer built atop a routing substrate that provides higher-level services. As we shall see, almost all of the software overhead attributed to various user communication services can be eliminated; applications need not sacrifice efficiency when using high-level communication services.

[3]Some systems may include buffer management to provide overflow safety. This only introduces additional buffer management overhead, but does not affect the other components.

4 Messaging layer with high-level network features

Several networks now provide higher level features. Examples include the Sunshine ATM switch [9] and DEC Autonet II [22] which provide in-order delivery of messages, and TRANSIT [17] which supports implicit acknowledgement of message reception. In this section, we consider the implementation of the protocols described in Section 3 using a messaging layer designed on top of a specific routing substrate which provides similar higher level services.

We consider the design of a messaging layer atop a low-cost routing framework called *Compressionless Routing* [16] which provides the following features in hardware:

- *Order-preserving* transmission
- *Deadlock freedom* independent of packet acceptance guarantees
- *Fault-tolerant* transmission at the packet level

Compressionless Routing (CR) provides order-preserving transmission by ensuring that messages, issued in sequence from the sender, must begin to arrive at their destination prior to completely entering the network. Most networks guarantee deadlock freedom only under the assumption that each output eventually extracts all packets delivered to it. However, CR ensures deadlock freedom independent of such acceptance guarantees by providing a mechanism for releasing the resources of a message path whenever the possibility of deadlock is detected. This mechanism eventually frees up all network resources, allowing other messages in the network to make progress even if a destination node, having committed all its resources, cannot extract any messages from the network. Finally, CR provides fault-tolerant packet delivery by exploiting the acceptance of the last flit as an end-to-end acknowledgement.

To characterize the benefits of these higher-level hardware services, we assume that they are implemented on hardware similar to the CM-5 network described in Section 3.1 with a packet size of five words. We first describe how the three protocols of Section 3 are implemented using the above high-level features, and then compare their costs with those of the CMAM-based implementations.

4.1 Costs of communication protocols

Single-packet delivery The costs for sending and receiving a single packet are identical to the CMAM case, costing 20 instructions at the source and 27 instructions at the destination. These costs are fixed by the network interface, which is identical in the two cases. However, unlike the CMAM implementation, single-packet delivery now meets all requirements of user communication services: it is guaranteed to be order preserving, fault-free and not cause deadlock or buffer overflow.

Finite sequence, multi-packet delivery This protocol for transferring data from a source memory buffer to a destination memory buffer is shown in Figure 5 and consists of four steps: First, the sender breaks up the large user message into several packets and injects them into the network (Step

1). Second, on successful receipt of the header packet which contains information about the size of the transfer,[4] the destination node allocates a buffer large enough to store the entire message being transmitted (Step 2). The destination then takes packets from the network and stores their data into the buffer associated with the transmission. The arrival of the last packet invokes the user handler (Step 3). This protocol differs from the corresponding CMAM-based protocol in three ways: there are no buffer allocation messages, no overhead for in-order delivery, and no end-to-end acknowledgements. Buffer preallocation is unnecessary since CR allows nodes, which have committed all their resources, to reject incoming messages (by rejecting the header packet) without deadlocking the network. In-order delivery comes for free since the CR routing substrate preserves transmission order, and the end-to-end acknowledgement is made unnecessary because each packet is reliably delivered.

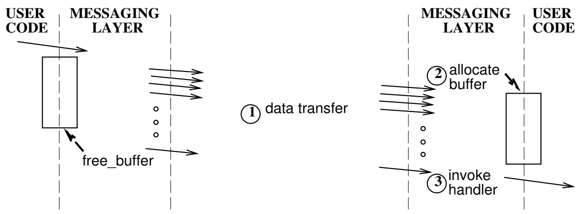


Figure 5: *Finite sequence protocol using high-level network features.*

Thus, there are only two components to the protocol cost: base single-packet deliveries (Steps 1 and 3) and buffer management (Step 2). The base cost consists of multiple transmissions of single packets. Buffer management has negligible overhead when compared with the CMAM-based protocol, its cost arising from having to store the pointer to the allocated buffer in a table, associating it with the incoming message.[5] Note that an additional advantage which does not show up in our accounting is that the user message need not be source buffered because data is reliably transmitted once it is successfully injected into the network.

The protocols costs are shown on the left in Figure 6. The bar charts compare the costs in the CMAM implementation (left bars) with the implementation using the high-level network features (right bars). The slightly lower destination cost in the high-level feature based protocol is due to fewer branches in the packet reception code, and a specialized last-packet handler. The costs of the high-level feature based protocol correspond exactly to the base costs of the CMAM implementations, providing a 10-50% improvement (based on message size) by effectively eliminating all the overheads of providing user communication requirements in software.

Indefinite sequence, multi-packet delivery The implementation of this protocol is simplified considerably with the higher level of network services offered by the routing substrate. This protocol is shown in Figure 7 and is implemented essentially for free on top of multiple single-packet transmissions. Unlike the corresponding CMAM-based protocol, there are no overheads for in-order delivery or fault-tolerance; both are supported directly in hardware.

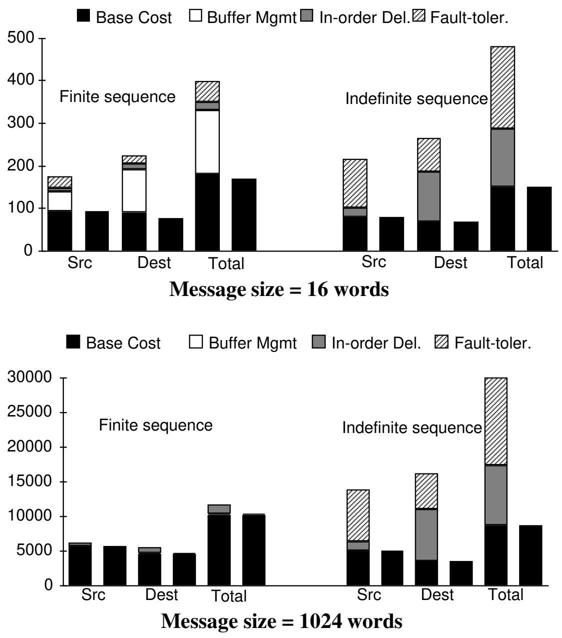


Figure 6: *Comparison of messaging layer costs.*

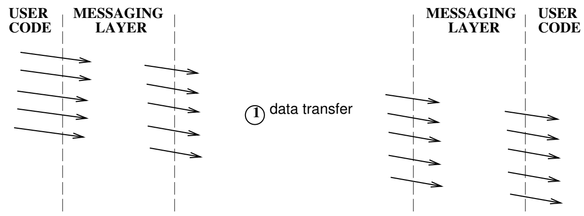


Figure 7: *Indefinite sequence protocol using high-level network features.*

The protocol costs are shown on the right in Figure 6. As can be seen from the comparison with the CMAM implementation, the higher-level network features reduce the software costs in the messaging layer by $\sim$70%.

We have seen in this section that the software cost of messaging can be significantly reduced by designing routing networks which replace (or otherwise make unnecessary) software to implement user communication requirements. Specifically, networks which provide message ordering, end-to-end flow control, and hardware support for fault-tolerance can significantly reduce the end-to-end cost of communication.

5 Discussion

No single study can address all of the issues. In this study, we have chosen to put aside issues of protection and

[4] We assume that the network interface can selectively accept a header packet based on a hardware check for the availability of node resources. A rejected header packet results in the tear down of the message path and later retransmission.

[5] As in the CMAM-based protocol, we exclude the actual allocation cost since our interest is only in the protocol costs.

Finite sequence, multi-packet delivery

Feature	Source Cost	Destination Cost	Total Cost (insns)
Base Cost	$3 + (18 + n)p$	$18 + (14 + n)p$	$21 + (32 + 2n)p$
Buffer Mgmt.	47	101	148
In-order Del.	$2p$	$1 + 3p$	$1 + 5p$
Fault-Toler.	27	20	47
Total	$77 + (20 + n)p$	$140 + (17 + n)p$	$217 + (37 + 2n)p$

Indefinite sequence, multi-packet delivery

Feature	Source Cost	Destination Cost	Total Cost (insns)
Base Cost	$(18 + \frac{n}{2})p$	$13 + (12 + \frac{n}{2})p$	$13 + (30 + n)p$
Buffer Mgmt.	–	–	–
In-order Del.	$5p$	$29p$	$34p$
Fault-Toler.	$(27 + \frac{n}{2})p$	$20p$	$(47 + \frac{n}{2})p$
Total	$(50 + n)p$	$13 + (61 + \frac{n}{2})p$	$13 + (111 + \frac{3n}{2})p$

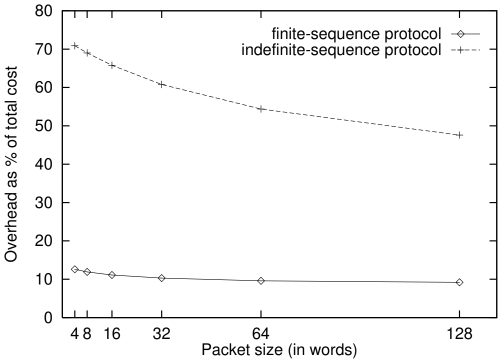


Figure 8: (**Left**) *Generalized breakdown of CMAM costs into messaging layer features:* n *= packet size in words,* p *= number of packets in message.* (**Right**) *Messaging layer overhead versus packet size for 1024 words of communication.*

processor scheduling, and focus on basic delivery, ordering, buffer management, and fault tolerance. Clearly, any solution which reduces messaging overhead to the level of tens of instructions must deal with the former issues.

Studying a specific 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 that 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 how our results would be affected by modest changes in machine architecture.

Larger packet sizes The CMAM implementation supports four data words per packet; however, other parallel machines and even the CM-5E network interface support larger packet sizes. Our results can be parameterized based on the hardware packet size, n (words per packet) and p (packets per message). These generalized costs are shown in Figure 8.

The parameterized results show that the CMAM messaging overhead is not just an artifact of the CM-5's small packet size. For example, the plot on the right of Figure 8 shows the messaging overhead for a 1024-word message as a fraction of the total software communication cost as the packet size is varied from 4–128 words. It is clear that messaging overhead for indefinite-sequence multi-packet delivery remains significant over the range of packet sizes. For finite-sequence multi-packet deliveries, the messaging overhead is lower, but still significant, accounting for 9–11% of the total cost.

Improved network interfaces and DMA hardware If network interfaces can be integrated on-chip, as in [12, 6], the basic cost of communication can be reduced, but this will not reduce protocol costs in the messaging layer on which our study focuses. If the base cost is reduced, that increases the importance of the costs in the rest of the messaging layer. Similarly, while DMA hardware can reduce the cost of moving large amounts of data, it is unlikely that it would give much benefit for the packet sizes we have considered. Further, as with network interface improvements, this would also reduce the base cost, increasing the importance of the software messaging layers.

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 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 large software costs. For example, a number of designs have proposed out-of-order delivery in order to improve network routing performance (randomization [18] or adaptive routing [20, 5, 7]). Our results show clearly that packet sequencing and reordering incur a significant cost, so the benefits of out-of-order delivery for the network must be weighed against the software costs of such behavior. Because software overhead is generally much larger than 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. As we have shown, a network design which provides some simple high-level features can eliminate the need for parts of the messaging layer, reducing communication cost. Specifically, a network that provides in-order delivery, end-to-end flow control, and reliable delivery would obviate much of the need for a messaging layer, reducing the messaging cost to that of basic data movement.

Reducing communication features We have focused on how to support high level communication features, but an obvious alternative is to provide lower level services to the application. Such an approach has the drawback that it places the burden on the parallel software to build messaging services and manage their cost as appropriate. This choice is problematic as software is already a major challenge in most parallel systems. Achieving low cost implementations without compromising on functionality has the significant advantage of insulating programmers and compilers from reasoning about network scheduling and resource usage.

Communication cost versus latency Finally, we have focused on instruction counts as the primary measure of communication cost. 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 hardware 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 primary distinction of our study is that it 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 [23, 8]. 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. Only recently have routing studies begun to consider the possibility of supporting higher level communication features in hardware to increase end-to-end performance [22, 16].

Research on network interfaces has focused primarily on reducing message injection (and reception) overhead [12, 6, 19] or offloading the communication onto a coprocessor [13, 15, 2]. Improvements in network interfaces only reduce the basic communication cost. From our studies, we have seen that 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 several 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 [1, 21] 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.[6] While reduced functionality layers may be a necessary expedient, a better long term solution would involve full functionality communication primitives and high performance.

7 Conclusion

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. Our study provides a piecewise breakdown of the communication costs, relating specific network features to the software components required for supporting user communication services. Our results show that up to 50–70% of end-to-end messaging cost is attributable to the arbitrary order of packet delivery, finite buffering, and limited fault-handling capabilities. For example, despite CMAM's efficiency, the cost of delivering a 16-word message is between 285 and 481 instructions using a finite sequence, multi-packet protocol. While improvements to the CM-5 network interface are possible, paradoxically, such improvements will only worsen the situation, causing these network features to contribute an even larger percentage of the cost.

One way to reduce these software overheads is by using messaging layers which run atop networks providing higher-level services. Networks similar to Compressionless Routing which provide services such as in-order delivery, end-to-end flow control, and fault-tolerance at the packet level are not only practical but also considerably simplify the design of software messaging layers. Our studies, using a messaging layer based on Compressionless Routing, show that virtually all software overheads associated with protocols atop lower-level networks can be eliminated.

This study highlights a fundamental tension 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 should also consider how their decisions affect the software messaging layers. Further, based on the results of this study, we believe the most significant way to further reduce messaging overhead is for the hardware to provide functionality which eliminates much of the need for software layers.

Acknowledgements

The research described in this paper was supported in part by NSF grant CCR-9209336, ONR grants N00014-92-J-1961 and N00014-93-1-1086, and NASA grant NAG 1-613.

The authors thank Jae Hoon Kim and John Plevyak for participating in several discussions about the messaging protocols described in the paper. We also thank the anonymous reviewers for their useful comments.

References

[1] 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.

[2] M. A. Blumrich et al. Virtual memory mapped network interface for the SHRIMP multicomputer. In *Proceeding of the International Symposium on Computer Architecture*, April 1994.

[3] V. Cerf and R. Kahn. A protocol for packet network interconnection. *IEEE Transactions on Communications*, 1974.

[4] W. J. Dally. Virtual channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, 1992.

[5] W. J. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):466–74, April 1993.

[6] W. J. Dally et. al The J-Machine: A fine-grain concurrent computer. In *Information Processing 89, Proceedings of the IFIP Congress*, pages 1147–1153, August 1989.

[7] J. Duato. On the design of deadlock-free adaptive routing algorithms for multicomputers: Design methodologies. In *Proceedings of Parallel Architectures and Languages Europe*, 1991.

[6] The CMAM round-trip protocol using the two separate CM-5 networks however is safe.

[8] C. Leiserson et al. The network architecture of the Connection Machine CM-5. In *Proceedings of the Symposium on Parallel Algorithms and Architectures*, 1992.

[9] J. Giacopelli et al. Sunshine: A high-performance self-routing broadband packet-switch architecture. *IEEE J. Selected Areas Comm.*, 9(8):1289 – 1298, October 1991.

[10] Message Passing Interface Forum. The MPI message passing interface standard. Technical report, University of Tennessee, Knoxville, April 1994.

[11] 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.

[12] 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.

[13] M. Homewood and M. McLaren. Meiko CS-2 interconnect Elan – Elite design. In *Proceedings of the IEEE Hot Interconnects Symposium*. IEEE TCMM, August 1993.

[14] B. Horst. Massively-parallel systems you can trust. In *Proceedings of COMPCON Spring '94*. IEEE Computer Society, IEEE Press, February 1994.

[15] Intel Corporation. *Paragon XP/S Product Overview*, 1991.

[16] 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.

[17] T. F. Knight. Technologies for low latency interconnection switches. In *Proceedings of ACM Symposium on Parallel Algorithms and Architectures*, 1989.

[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] M. D. Schroeder and M. Burrows. Performance of Firefly RPC. *ACM Transactions on Computer Systems*, 8(1):1–17, February 1990.

[22] Jim Scott et al. Link by link, per VC credit based flow control. Technical Report 94-0168, The ATM Forum Technical Committee, 1994.

[23] 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.

[24] Thinking Machines Corporation. *Connection Machine CM-5, Technical Summary*, November 1992.

[25] Thinking Machines Corporation. *CMMD Reference Manual, V3.0*, May 1993.

[26] 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.

A Breakdown of instruction counts for CMAM-based multi-packet protocols

The instruction counts for the multi-packet messaging protocols described in Section 3 can be further classified into several subcategories. We identify three subcategories based on the cost hierarchy prevalent in existing machines:

1. register-based instructions (**reg**)
2. loads and stores to memory (**mem**)
3. loads and stores to memory-mapped devices (**dev**)

It is expected that instructions belonging to the **reg** category will incur less overhead than instructions belonging to the **mem** and **dev** categories. Such a classification enables the messaging overhead to be characterized in terms of cycle counts using a simple weighted cost model. These models can in turn be specialized to the machine and applications of interest. For example, a model for the CM-5 hardware might assume that **reg** and **mem** instructions cost 1 cycle each, while a **dev** instruction costs 5 cycles [24].

Table 3 shows the instruction count breakdown into the subcategories above for the CMAM-based multi-packet protocols described in Section 3.

Message size = 16 words

Finite sequence, multi-packet delivery

Feature	Source			Destination		
	reg	mem	dev	reg	mem	dev
Base Cost	62	9	20	62	11	17
Buffer Mgmt.	36	1	10	79	12	10
In-order Del.	8	–	–	13	–	–
Fault-toler.	22	–	5	14	1	5
Total	128	10	35	168	24	32

Indefinite sequence, multi-packet delivery

Feature	Source			Destination		
	reg	mem	dev	reg	mem	dev
Base Cost	56	4	20	52	–	17
Buffer Mgmt.	–	–	–	–	–	–
In-order Del.	8	12	–	70	46	–
Fault-toler.	88	8	20	56	4	20
Total	152	24	40	178	50	37

Message size = 1024 words

Finite sequence, multi-packet delivery

Feature	Source			Destination		
	reg	mem	dev	reg	mem	dev
Base Cost	3842	513	1280	3086	515	1025
Buffer Mgmt.	36	1	10	79	12	10
In-order Del.	512	–	–	769	–	–
Fault-toler.	22	–	5	14	1	5
Total	4412	514	1295	3948	528	1040

Indefinite sequence, multi-packet delivery

Feature	Source			Destination		
	reg	mem	dev	reg	mem	dev
Base Cost	3584	256	1280	2572	–	1025
Buffer Mgmt.	–	–	–	–	–	–
In-order Del.	512	768	–	4480	2944	–
Fault-toler.	5632	512	1280	3584	256	1280
Total	9728	1536	2560	10636	3200	2305

Table 3: *Instruction subcategories for CMAM-based finite sequence and indefinite sequence, multi-packet protocols.*