Cut-Through Delivery in Trapeze: An Exercise in Low-Latency Messaging[*]

Kenneth G. Yocum Jeffrey S. Chase Andrew J. Gallatin Alvin R. Lebeck
Dept. of Computer Science
Duke University
Durham, NC 27708-0129
{grant, chase, gallatin, alvy}@cs.duke.edu

Abstract

New network technology continues to improve both the latency and bandwidth of communication in computer clusters. The fastest high-speed networks approach or exceed the I/O bus bandwidths of "gigabit-ready" hosts. These advances introduce new considerations for the design of network interfaces and messaging systems for low-latency communication.

This paper investigates cut-through delivery, a technique for overlapping host I/O DMA transfers with network traversal. Cut-through delivery significantly reduces end-to-end latency of large messages, which are often critical for application performance.

We have implemented cut-through delivery in Trapeze, a new messaging substrate for network memory and other distributed operating system services. Our current Trapeze prototype is capable of demand-fetching 8K virtual memory pages in 200μs across a Myrinet cluster of DEC AlphaStations.

1. Introduction

Advances in network technology continue to improve the latency and bandwidth of communication in computer clusters. The tighter coupling of cluster nodes creates new opportunities to reduce the running time of large-scale computations by using hardware resources across the cluster in a coordinated way.

[*]This work is supported in part by NSF grant CDA-95-12356, Duke University, and the Open Software Foundation. Jeff Chase is partially supported by NSF Career Award CCR-96-24857. Alvin Lebeck is partially supported by NSF Career Award MIP-97-02547

Latency of network communication is an important factor in determining the effectiveness of cluster-based parallel computing, distributed file storage, network memory, and other resource sharing schemes. Network hardware and software are typically optimized for high-bandwidth continuous data transfer or low-latency exchanges of *small* messages of a few hundred bytes. However, in many instances, network communication involves *large* messages on the order of one to ten kilobytes. Latency of large messages is critical for page migration, block data transfer for parallel applications, or demand fetching of virtual memory pages or file blocks.

This paper describes the design and implementation of *Trapeze*, a new messaging system that delivers low latency for both large and small messages. Trapeze was designed primarily to handle page migration traffic in the Global Memory Service (GMS), a cooperative memory system for clusters. The Trapeze prototype consists of a messaging libary and custom firmware for Myrinet, a high-performance cluster interconnect. GMS and the Myrinet platform are described in Section 2.

Trapeze uses several important buffering and DMA management optimizations to minimize the latency of large messages, in this case page transfers. Many of the optimizations used by Trapeze have been used in earlier fast messaging implementations. This paper gives an overview of Trapeze and focuses on a principal technique not described or evaluated elsewhere—called *cut-through delivery*. Cut-through delivery minimizes latency by aggressively overlapping the four DMA transfers needed to move a large message from one host to another across a network (Figure 1): (1) sender's host memory to adapter, (2) sender's adapter to network link, (3) network link to receiver's adapter, and (4) receiver's adapter to host memory.

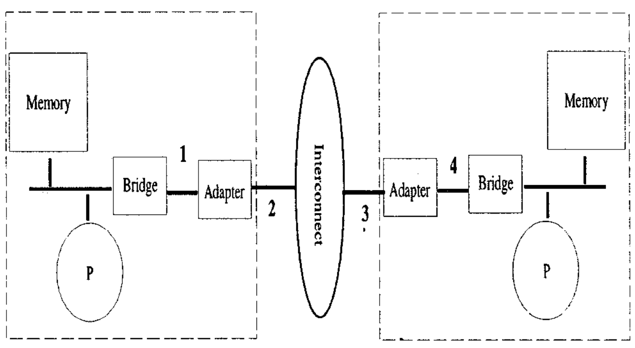


Figure 1. Four DMA Transfers for a Network Message

Cut-through delivery eliminates network adapter store-and-forward latency by pipelining packets through the adapter, similar to the way *cut-through switching* [13] eliminates store-and-forward latency in high-performance network switches. With cut-through delivery, large messages flow *through* the network adapter; the adapter can place data at the sink port shortly after it arrives at the source, without waiting for the entire packet to be transferred onto the adapter. In this way, the DMA transfers from the source and to the sink are overlapped, on both the sending and receiving sides.

We do not claim that cut-through delivery is novel. Indeed, variants of cut-through delivery are used in low-cost network interfaces to reduce the need for expensive buffer space on the adapter. Instead, we argue that cut-through delivery is a fundamental issue in network interface design that has not been adequately explored. We show that cut-through delivery significantly reduces large-packet latencies, and that the expected benefit of cut-through delivery grows rapidly as the network link speed approaches I/O bus speeds, as is the case with modern high-speed networks. In our cluster, cut-through delivery reduces 8KB page transfer latencies by 43% (to 177 us) after all other known optimizations have been applied. As technology advances, high-speed networks will drive I/O bus design, maintaining the relevance of this optimization. Finally, cut-through delivery is missing from other fast messaging systems we have seen.

This paper is organized as follows. Section 2 presents the motivation and background for low-latency page transfers, and sets Trapeze in context with other fast messaging systems. Section 3 outlines the implementation of cut-through delivery in Trapeze for Myrinet. Section 4 presents microbenchmark results to evaluate the Trapeze prototype on a Myrinet/Alpha cluster. We conclude in Section 5.

2. Overview of Trapeze

Trapeze was designed as a high-performance communication substrate for cluster operating system services such as cooperative virtual memory, rather than as a full-featured messaging system. This role has dictated the features of Trapeze and the character of the messaging interface. However, cut-through delivery and other optimizations used in Trapeze are applicable to any messaging system or application that is sensitive to large-message latency, including some parallel applications built using message passing (e.g., MPI [11]) or software distributed shared memory.

In this section we (1) discuss the importance of large-message latency for a specific system that uses Trapeze, (2) give an overview of Myrinet and the Trapeze messaging system, and (3) relate Trapeze to other low-latency messaging systems.

2.1. Importance of Latency for Network Memory

Our first use of Trapeze is as a messaging substrate for the *Global Memory Service* (GMS) [9], a Unix kernel facility that manages the memories of cluster nodes as a shared, distributed page cache. The GMS implementation supports remote paging [5, 10] and cooperative caching [6] of file blocks and virtual memory pages, unified at a low level of the operating system kernel.

The purpose of GMS is not to support a shared memory abstraction, but rather to transparently improve the performance of data-intensive workloads. GMS coordinates memory usage across the cluster so that nodes can satisfy paging and file operations with memory-to-memory network transfers whenever possible, avoiding disk accesses. The key insight is that improvements in disk latency are limited by mechanical factors, whereas network performance has improved at a rapid rate. For example, even with a 100 Mb/s Ethernet interconnect, a 266 MHz AlphaStation 500 running a GMS-enhanced Digital Unix 4.0 kernel can demand-fetch a file block or virtual memory page from the memory of a peer in 1.2 ms, an order of magnitude faster than the average 12 ms access time of a local fast/wide Seagate Barracuda disk.

However, the performance of data-intensive applications under GMS is still dominated by communication latency. GMS may reduce I/O stall times by an order of magnitude or more, but a 266 MHz Alpha 21164 CPU could issue up to a million instructions in a millisecond spent idling for a remote page fault. Moreover, experience with GMS on several networks has shown that large message latencies are often higher than expected. High bandwidth is most easily achieved with continuous streams of packets that naturally pipeline in the network and adapters, but this does not translate into low-latency delivery of individual large messages sent as a single packet. For example, a 155 Mb/s ATM network can in principle deliver an 8K message in under 400 μs. In practice, the original GMS prototype measured page transfer times above a millisecond using high-quality ATM adapters. The GMS developers have explored pipelined subpage fetches [12] and other approaches to masking fetch latency.

2.2. Page Transfers on Myrinet

Our approach to minimizing page transfer costs is to use a custom messaging system for Myrinet [3], a high-speed wormhole-routed LAN. Our Myrinet configuration consists of 8-port crossbar switches and adapters (Network Interface Cards or NICs) that attach to our AlphaStation hosts through the 32-bit PCI I/O bus.

The Myrinet NICs are programmable, providing a flexible interface to the host. The NICs include a 256K block of dual-ported static RAM and a custom CPU and link interface based on the third-generation LANai 4.1 chipset. The NIC SRAM is addressable from the host physical address space; the host and LANai interact by reading and writing locations of this shared memory. The behavior of the adapter is determined by a firmware program (a Myrinet Control Program or MCP) that is written into the NIC SRAM from the host at startup. Myrinet messaging latencies are determined primarily by overhead in the MCP and host messaging software and the time to transfer data on the host I/O bus.

Although Myrinet can handle a large transfer as a single packet, experiments using the vendor-supplied firmware (supporting a host message interface called *MyriAPI*) showed that latency grew more steeply with message size than we had expected. A one-way 8K page transfer took over 400μs at best, almost a factor of three higher than the minimum achievable latency. Sending page transfers as Internet datagrams over *MyriAPI* yielded demand-fault times of over 850μs in the best case, even with a well-optimized network driver.

We investigated other message systems available for Myrinet in the research community, e.g., Active Messages (AM) [17] and Fast Messages (FM) [15]. Like *MyriAPI*, these systems support full-featured APIs designed primarily to meet the needs of parallel applications. While they reported superior performance for small messages, they did not support the DMA facilities we needed for zero-copy page fetches, relied on polling for detection of received messages, and were not available for LANai 4.1 or Alpha-based hosts. Finally, where large-message latency timings were reported, they were comparable to *MyriAPI*. Page transfers and other large messages can be sent using bulk transfer extensions (e.g., FM's "streaming messages" interface), which fragment transfers into pipelined packet streams, but this did not meet our goal of transferring a memory page and associated control information as a single packet with the lowest possible latency and overhead.

2.3. An Overview of Messaging with Trapeze

Our solution was to develop Trapeze, a simple, raw transport layer designed to provide low latency for both large and small messages. The Trapeze prototype consists of custom Myrinet firmware (Trapeze-MCP) and a host library that implements the messaging interface (Trapeze-API). To allow kernel-kernel communication, the host library is linked into the Digital Unix 4.0 kernel, which includes a driver to initialize the device and to field interrupts. The Trapeze-MCP manages the network link, coordinates message buffering and DMA, and optionally generates host interrupts for incoming packets.

The dominant design goal of Trapeze is to support low-latency, zero-copy transfers of virtual memory pages across the interconnect. Our current implementation is capable of demand-fetching 8K pages with latencies below 200 μs, about fifty times faster than the average access time for a high-quality disk. Trapeze implements several optimizations to achieve this goal. In this paper we limit our attention to the cut-through delivery technique, which pipelines individual large messages within the adapters, transparently to the hosts.

Focusing on page transfers allowed us to make several simplifications to streamline the messaging system and reduce our prototyping effort:

- **Fixed-size message and payload buffers.** A Trapeze message is a 128-byte *control message* with an optional attached *payload* of up to 8KB. Control message buffers are contiguous, aligned blocks of adapter memory; payload buffers are frames of host physical memory.
- **No protection.** Trapeze currently supports only a single logical communication endpoint or channel, intended for but not limited to kernel-to-kernel

communication. We assume that the interconnect is secure and the hosts are trusted.

- **Best-effort delivery**. Packets are never dropped by the Myrinet network itself, which provides link-level flow control by backpressure from a bottleneck interface. However, to avoid deadlocking the interconnect, the MCP will drop received packets if the host fails to keep up with incoming traffic on the link. It is the responsibility of the message sender and receiver to coordinate end-to-end flow control and recovery from dropped messages, if any is needed.

Table 1 lists the Trapeze-API routines used for the experiments in this paper. These routines interact with the Trapeze-MCP through an endpoint structure containing a pair of buffer rings: a *send ring* for outgoing messages and a *receive ring* for incoming messages. Each ring is an array of 128-byte control message buffers in the NIC SRAM, managed as a circular producer/consumer queue. Each ring entry includes space for the message contents and header fields for control information.

The host Alpha processor accesses control message contents directly using programmed I/O. The Trapeze-API `tpz_get_sendmsg` and `tpz_get_rcvmsg` routines each return a pointer (`msg_t`) to a ring entry; the application (e.g., the GMS kernel module) has exclusive access to the buffer until it releases it with `tpz_release_msg`. The intent is that the caller moves message data directly between processor registers and the valid locations of the buffer.

Payload frames can be attached to entries in either ring. The Trapeze-API attaches a payload frame (`vm_page_t`) by storing the frame physical address into the ring entry in a form that the MCP can use to request DMA to or from the frame. If a payload is attached to an outgoing send ring entry, the MCP sends the payload contents along with the message. Frames attached to the receive ring are used as buffers for incoming payloads. An incoming control message is deposited in the next available receive ring entry; any payload is transferred via DMA to the frame attached to that entry, or discarded if no frame is available.

2.4. Related Messaging Systems

The basic structure of Trapeze is similar to AM, FM, and other fast messaging systems designed to minimize latency of small messages by eliminating operating system overheads and network protocol processing (e.g., Hamlyn [4], U-net [1], SHRIMP [2], and others). Our work was also influenced by the Osiris [8] and FRPC work [16], which identify adapter design issues for low-latency messaging on high-speed networks.

Some of these systems include bulk data transfer facilities optimized for high bandwidth, but none of the published work describes cut-through delivery optimizations or identifies low latency for large packets as an explicit design goal. The FRPC and Osiris platforms had I/O buses offering much higher bandwidth than the network links, thus large-packet latency was limited by link bandwidth. Trapeze is similar in spirit to its predecessors, but it provides more specialized functionality and an explicit emphasis on optimizations for large packets on modern high-speed networks.

3. Cut-Through Delivery in Trapeze

The Trapeze-MCP uses cut-through delivery to ensure maximum utilization of the network link and the PCI I/O bus when transferring message payloads using DMA. Cut-through delivery simply means that the MCP always initiates DMA as soon as possible, in order to maximize overlapping of the DMA transfers needed to move a packet between the link and the host.

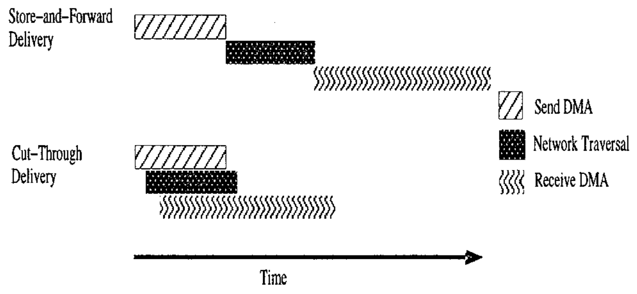


Figure 2. Cut-Through Delivery vs. Store-and-Forward on the AlphaStation 500

The MCP initiates DMA transfers by storing to LANai registers and then waiting for the DMA unit to signal completion by interrupting the LANai processor. There are four possible DMA operation types: host to NIC or NIC to host on the PCI bus, and link to NIC or NIC to link on the network interface. The Myrinet network link is bidirectional, but only one transfer at a time can take place on the PCI bus. Thus there are effectively three DMA functional units.

The Trapeze-MCP uses a resource-centered structure (Figure 3) to maximize utilization of the three DMA units. On each iteration through its main loop, the MCP

Operations on Message Rings and Slots	
msg_t tpz_get_sendmsg()	Allocate a send ring entry for an outgoing message; fail if no entry is available. The caller builds the message (including header) in the buffer with a sequence of **store** instructions.
msg_t tpz_get_rcvmsg()	Receive the next incoming message; fail if no message is pending. The caller reads the message contents with a sequence of **load** instructions.
msg_t tpz_release_sendmsg() msg_t tpz_release_rcvmsg()	Release a ring entry. On the send ring, this sends the message in the entry.
Payload Operations	
tpz_attach_sendmsg(msg_t,vm_page_t, io_completion_t,caddr_t)	Attach a payload frame to a ring entry. The frame contents shall be sent with the message as a payload, and the completion routine will be called, with its arguments, either when the send entry is re-used, or when tpz_check_xmit_complete(msg_t) is called.
tpz_attach_rcvmsg(msg_t,vm_page_t)	Attach a frame for a payload receive buffer to a receive ring slot.
vm_page_t tpz_detach_sendmsg(msg_t) vm_page_t tpz_detach_rcvmsg(msg_t)	Detach a payload buffer from a ring entry. Used to extract the source buffer for a send message, or to retrieve the payload received with an incoming message.
void tpz_check_xmit_complete(void)	Calls io_completion routines to free payload frames for transmitted packets.
void tpz_set_payload_len(msg_t,int) int tpz_get_payload_len(msg_t)	Set the length of the payload to be transmitted, or retrieve the payload length of a received packet.

Table 1. Trapeze Messaging API Subset

asks: *which of the three DMA engines are idle now, and how can they be used?*

The alternative functional view is best illustrated by the current LANai Active Messages prototype (LAM) [14]. LAM's sending and receiving sides execute as separate loops using a simple coroutine facility. Each iteration through the loop asks: *is the current packet ready for the next stage of processing?* If a coroutine detects that the previous step in processing a packet has not yet completed, it transfers control to the other coroutine using a special LANai **punt** instruction. The functional LAM structure achieves near-optimal pipelining of the DMA transfers for successive outgoing packets, but it does not provide any DMA overlap for individual packets on either the sending or receiving sides.

3.1. Cut-Through Delivery

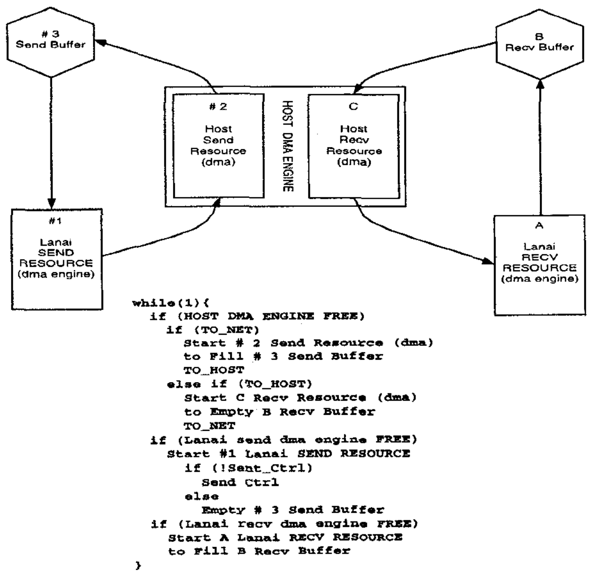
The resource-centered approach of the Trapeze-MCP enables *intra-packet* DMA pipelining, to reduce the latency of individual packets. The NIC is viewed as a cut-through device rather than a store-and-forward device, overlapping the transfer of the message across the sending host I/O bus, the network, and the receive host I/O bus. As shown in Figure 2, the pipelining of cut-through delivery can reduce message transfer time compared to store-and-forward delivery by hiding the latency of transfers on the host I/O bus, which is the bottleneck link in our configuration.

Cut-through delivery works for both incoming and outgoing packets as follows. On the sending side, the MCP initiates DMA of an outgoing packet onto the network link as soon as a sufficient number of bytes have arrived from host memory over the PCI bus. On the receiving side, the MCP initiates DMA of the incoming packet into host memory as soon as a sufficient number of bytes have been deposited in NIC SRAM from the network link. Cut-through delivery performs this pipelining on a single packet, rather than fragmenting into smaller packets and incurring additional packet handling overheads.

Cut-through delivery must be implemented carefully to prevent the DMA out of the adapter from overrunning the DMA into the adapter, on either the sending or receiving sides. The Trapeze MCP initiates the outgoing DMA as a series of shorter *pulses*. The MCP initiates an outgoing pulse as soon as a threshold number of incoming bytes (set by the *pulse threshold* parameter) have arrived and the outgoing DMA engine is available. The LANai exposes the status of active DMA transactions to the firmware through counter registers for each DMA engine; the Trapeze-MCP main resource loop queries these counters to trigger outgoing DMA pulses.

3.2. Discussion

The primary goal of cut-through delivery is to overlap I/O bus transfers with network transfers. This must be balanced against the bus cycles consumed to acquire the bus for a larger number of smaller transfers, which reduces the bus bandwidth available for transferring data. Historically, LANs achieved only a fraction of the I/O bus bandwidth, and cut-through delivery would have negligible effect on large message latency. Therefore,



```
while(1){
  if (HOST DMA ENGINE FREE)
    if (TO_NET)
      Start # 2 Send Resource (dma)
      to Fill # 3 Send Buffer
      TO_HOST
    else if (TO_HOST)
      Start C Recv Resource (dma)
      to Empty B Recv Buffer
      TO_NET
  if (Lanai send dma engine FREE)
    Start #1 Lanai SEND RESOURCE
      if (!Sent_Ctrl)
        Send Ctrl
      else
        Empty # 3 Send Buffer
  if (Lanai recv dma engine FREE)
    Start A Lanai RECV RESOURCE
    to Fill B Recv Buffer
}
```

Figure 3. Trapeze MCP Structure

high-end LAN adapters may use store-and-forward delivery to enable larger bus transfers and attain maximum bandwidth for streams of packets. This can result in higher latencies for large messages, e.g., page transfers sent as individual AAL5 frames over an ATM network.

Ethernet and ATM network adapter designs may employ a form of cut-through delivery to reduce cost. A common design includes a receive FIFO that can hold a small amount of data (e.g., a few ATM cells), which are deposited into host buffers with DMA as new data arrives. This eliminates the need for enough adapter memory to buffer an entire packet or Protocol Data Unit.

For Ethernet, cut-through delivery will not reduce latency significantly because the maximum transmission unit (MTU) is too small. For ATM adapters, cut-through delivery can overlap DMA of leading cells into the host with DMA of trailing cells into the adapter, reducing large packet latency. However, this may also reduce bandwidth. The tradeoff is determined by the size of the FIFO or by more sophisticated balancing of batching and cut-through optimizations [7]. For Trapeze, the balance can be adjusted by varying the pulse thresholds.

In the next section we quantify the effects of cut-through delivery on large packet latency and bandwidth, focusing on the following questions:

- How much does cut-through delivery improve large-packet latencies in a real system?
- How sensitive is the benefit of cut-through delivery to the DMA pulse sizes? How can the optimal pulse size for a specific platform be determined?
- What is the smallest message size that benefits from cut-through delivery?
- Will cut-through delivery continue to reduce large-packet latency as networks and I/O buses advance?

4. Performance Analysis

This section presents the measured performance of cut-through delivery on our platform, and analytically derives expected benefits on other platforms with different I/O bus speeds. We first evaluate the critical design tradeoffs for cut-through delivery, then describe and evaluate a refinement called *eager DMA* that dynamically adapts DMA transfer sizes in response to the observed behavior of the I/O bus and network link. We then present performance numbers for a complete demand fetch of a page from a remote host's memory, including request latency and interrupt-based notification.

The timings in this section are taken from microbenchmarks of Trapeze on DEC AlphaStation 500 workstations with 266 Mhz 21164 Alpha processors, 2MB of off-chip cache, 128MB of main memory and LANai 4.1 Myrinet adapters. The I/O bus is a 33MHz 32-bit PCI with a Digital 21171 PCI bridge.

4.1. Measured Benefits of Cut-Through Delivery

We first present timings for cut-through delivery using a range of pulse thresholds. The *send threshold* is the number of bytes that must arrive from the host before the sending NIC initiates DMA to the link. The *receive threshold* is the number of bytes that must arrive from the link before the receiving NIC initiates DMA to host memory.

The pulse thresholds are critical tuning parameters. Small thresholds result in smaller DMA transfers, which can decrease the effective bandwidth of the I/O bus due to the overhead of acquiring the bus for each transfer. On the other hand, higher thresholds yield smaller pipelining benefits for individual packets. Moreover, the optimal pulse threshold may depend on the relative speeds of source and sink links. Our goal is to empirically determine the optimal value for both pulse thresholds on our platform.

Figure 4 shows the effect of send and receive thresholds on one-way page transfer latency. For these experiments, the DMA pulse size was fixed at the pulse threshold. The left-hand graph plots the send threshold on the

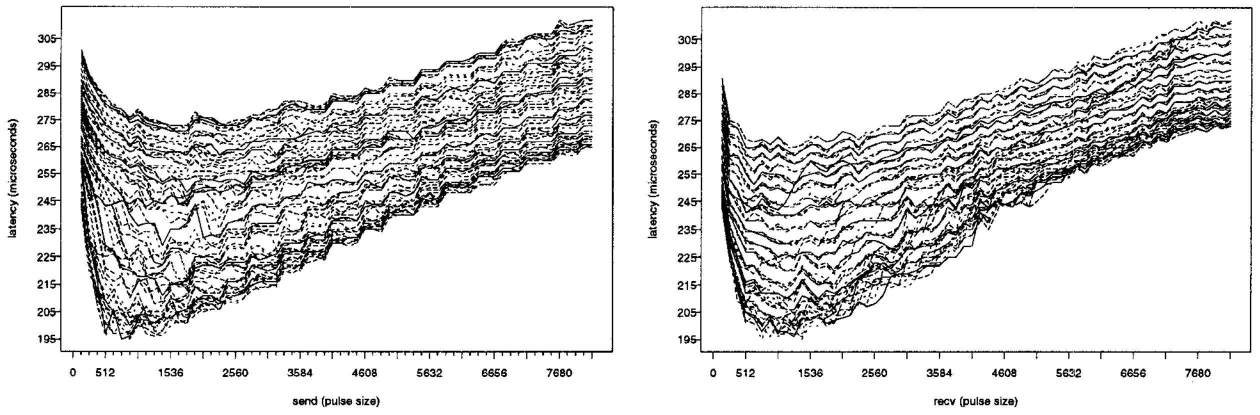


Figure 4. Impact of Pulse Threshold on Trapeze Latency for 8K Payloads

x-axis; each line represents a different value for the receive pulse. The right-hand graph shows the same data with the receive thresholds on the x-axis. These timings represent one half the average of 50 host-to-host round-trip page transfers, each consisting of a 128-byte control message and an attached 8KB payload. The hosts poll the LANai memory (with an optimal backoff determined empirically) to detect message arrival. We observed less than a 10μs variation from the average.

As expected, page transfer latency is highest at both ends of the graphs, due to limited overlap of large pulses and the DMA setup overhead of small pulses. The upper right point of each graph represents the latency for store-and-forward; both send and receive pulse sizes are 8KB, so there is no pipelining benefit. The lowest points correspond to the optimal pulse thresholds on the sending and receiving sides. Cut-through delivery with optimal thresholds reduces page transfer latency from 308μs to 195μs, a 37% improvement.

Note that the two graphs have different shapes, indicating that optimal pulse sizes may not be equal on the sending and receiving sides. We believe this is due to an imbalance in the PCI performance of our platform. This imbalance is readily seen from Figure 5, which shows the DMA bandwidth for transferring data to and from the I/O bus as a function of DMA sizes ranging from 64 bytes to 8192 bytes (the base VM page size in Digital Unix 4.0). The timings were produced by Myricom's *hswap* MCP, using a cycle counter on the adapter. Peak DMA read bandwidth (transfers from main memory to the NIC) approaches 128 MB/s, the maximum achievable by the 32-bit PCI standard. However, the DMA write bandwidth (transfers from the NIC to main memory) is about half the read bandwidth, apparently due to a flaw in the 21171 PCI bridge. With these numbers (and

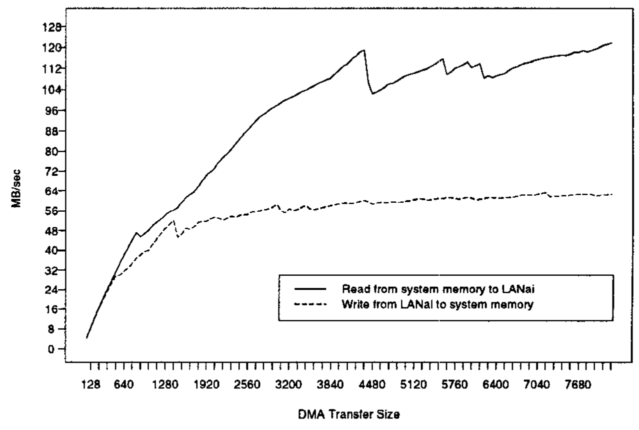


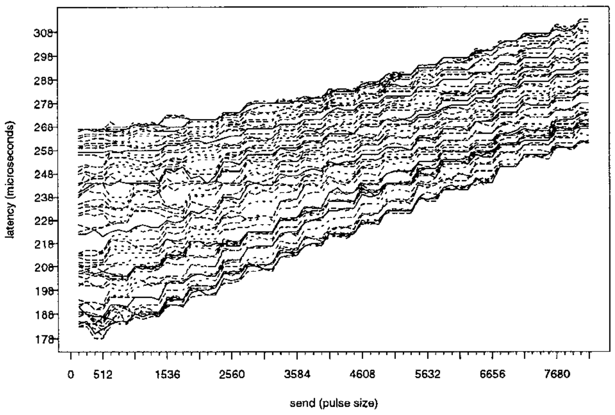
Figure 5. DEC Alphastation 500 DMA performance

assuming a gigabit network link), a store-and-forward transfer should take no less than 256μs, plus control message latency. This is consistent with the 308μs using Trapeze with store-and-forward delivery.

Although the I/O DMA bandwidth profile and imbalances may require different send and receive thresholds on some systems, on the 500 platform a single pulse size of 1280 bytes is near optimal. However, latency increases rapidly for adjacent pulse values.

4.2. Eager DMA

The upturn in the left-hand side of the graph in Figure 4 suggests a refinement to cut-through delivery. The upturn shows that although smaller pulse sizes extract the maximum benefit from pipelining, the overhead of the additional DMA transfers overshadows this effect. This is because the amount of data transferred with each



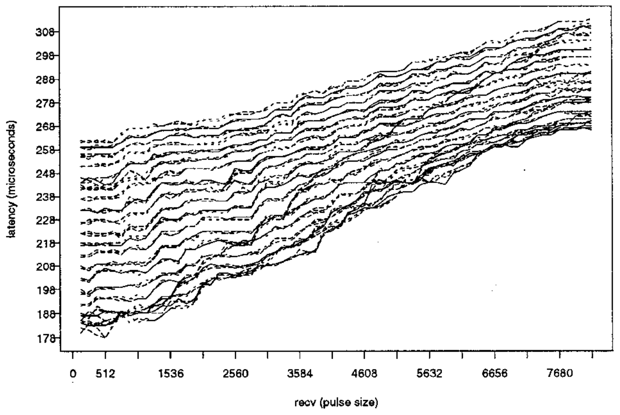


Figure 6. Trapeze One-Way Page Transfer Latency with Eager DMA (128 + 8K bytes)

pulse is fixed to the pulse threshold, even though more data may have arrived from the source by the time the sink DMA engine is idle.

We enhanced the Trapeze-MCP to combine the benefits of small and large thresholds using a technique we call *eager DMA*. Eager DMA treats the pulse threshold as a *minimum* unit of transfer, but moves all the data available at the start of each DMA transfer. For example, on the receiving side, the MCP initiates DMA to the host as soon as the number of bytes received from the network link exceeds the threshold. When the host DMA pulse completes, the MCP checks the number of bytes that arrived from the network while it was in progress. If this amount exceeds the pulse threshold, the MCP immediately initiates a new DMA for *all* of the newly received data. Eager DMA achieves maximum transfer overlap with the smallest number of transfers.

Figure 6 shows the performance of the enhanced Trapeze firmware, using the same experiment as in Figure 4. As expected, eager DMA delivers lower latency than static pulse sizes. The effect is most pronounced for small pulse thresholds, allowing Trapeze to achieve an average one-way page transfer latency of 177μs with any of the following send/receive pulse threshold pairs: (384/512), (512,512), (512, 384). This is an additional 9% improvement over the previous minimum of 195μs, for an overall 43% improvement over store-and-forward.

4.3. Effect of Payload Size

Although the previous analysis measured 8KB page transfers, cut-through delivery can reduce transfer latency for messages as small as 1KB. Figure 7 shows message latency as a function of payload size for both cut-through delivery and store-and-forward delivery. The benefit of cut-through delivery starts at 1KB payloads, reducing latency to 63μs compared to 68μs for store-and-forward. The relative advantage of cut-through delivery increases with payload size, with a 43% latency reduction for 8KB messages.

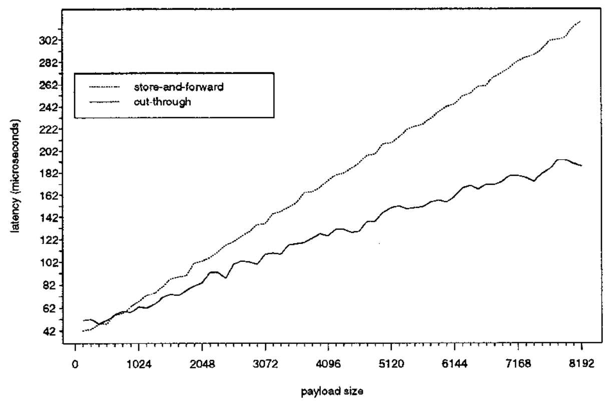


Figure 7. Cut-Through vs. Store-and-Forward Varying Payload Size

4.4. Effect of I/O Bus Bandwidth

With optimal cut-through delivery, large-packet latency is determined by the time to move the packet across the bottleneck link. Since the network link DMAs are overlapped even with store-and-forward, cut-through delivery is most effective when the network link bandwidth approaches or exceeds the available I/O bus bandwidth. In this case, bus transfer time is a significant component of total latency using store-and-forward delivery. This is the case with Myrinet on our AlphaStations: the link bandwidth is 1.28 Gb/s in both directions, while the I/O bus can transfer in only one direction at a time, achieving read bandwidth equal to the link bandwidth and write bandwidth only half that. On this platform, up to 75% of the total store-and-forward latency is bus transfer time, mostly on the receiving side.

Figure 8 shows the expected speedup of an 8KB page transfer as a function of the bandwidth ratio, $I/OBandwidth/NetworkBandwidth$, with network bandwidth fixed at a gigabit. The y axis is the speedup of cut-through delivery over store-and-forward, relative to a base configuration with network bandwidth and host DMA read and write bandwidths of 128MB/s. Note that this graph ignores fixed overheads, e.g., the time to set up a DMA transfer or to propagate the first byte through the network.

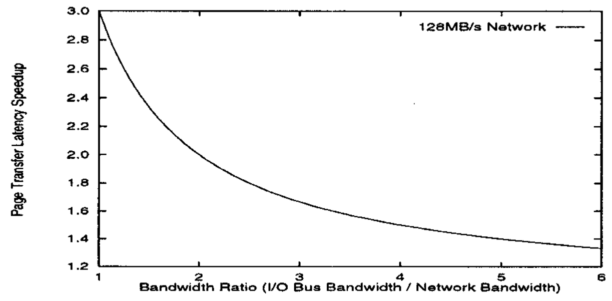


Figure 8. Expected Impact of Bandwidth Ratio

Figure 8 indicates that the expected speedup of optimal cut-through delivery is almost a factor of three if the bandwidths are perfectly matched. This is because the transfer latencies are equal across all three links (the sender and receiver I/O buses and the network link). This is the speedup expected from cut-through delivery using current Myrinet adapters on a top-quality 32-bit PCI bus. The expected benefit drops to a factor of two if the I/O bus bandwidth is double the network bandwidth, and to 33% if the bus is four times the link speed.

While these expected speedups are optimistic in that they ignore overhead — and therefore overestimate the percentage of store-and-forward latency that can be hidden by pipelining — they are pessimistic in that they assume the full I/O bus bandwidth is available to the network adapter. Even ignoring bus contention from competing devices, the adapter must divide the bus between incoming and outgoing transfers. This effectively cuts the bus bandwidth in half for a NIC that is sending or receiving at full speed while delivering a latency-critical packet in the other direction.

4.5. Putting it All Together: Demand Fetch

One-way page transfer is only one aspect (although an important one) of network performance for the Global Memory Service and other distributed operating system services. These systems use short messages for status updates or requests. For example, GMS uses short messages to update maps of the global page cache and to request a page when a fault occurs.

Average one-way latency for a typical short Trapeze control messages (without payload) is 27μs. This includes all sender and receiver overhead for processing the message, including access to 32 bytes of the buffer on the sending and receiving sides. Since a demand fetch includes one control message for the request and one page transfer for the response, we expect a demand page fetch with Trapeze to take approximately 205μs, close to the measured average of 201μs to demand-fetch an 8K page. This is a 35% improvement over store-and-forward fetch, which requires 310μs on average.

While cut-through delivery can be used in general purpose messaging systems, Trapeze is designed to be kernel resident. In particular, the host code that services requests for page transfers executes in an interrupt handler. On our platform, the use of interrupts for message arrival notification has negligible effect on measured latencies. We have measured one-way page transfer and demand fetch latency of 177μs and 207μs respectively using interrupt notification for each message on the receiver, compared to 178μs and 201μs with polling. Although the average latency to field an interrupt is about 6μs on our platform, use of interrupts also reduces memory contention in the LANai by eliminating the need for spin-polling from the host in our microbenchmarks.

5. Conclusions

This paper discusses the implementation and measurement of cut-through delivery optimizations in Trapeze, a new messaging system for Myrinet networks. Cut-through delivery overlaps the DMA operations involved in transferring individual large messages from one host to another. Our results show that this pipelining technique can cut large message latency almost in half on our AlphaStation/Myrinet platform.

Two important factors determine the amount of intra-packet overlap achieved with cut-through delivery: (1) DMA pulse threshold, and (2) the I/O bus bandwidth profile and ratio of host I/O bus bandwidth to network bandwidth. The experimental results illustrate the effect of a range of pulse thresholds on 8K message latency on our platform. Refining cut-through delivery, with *eager DMA* allows use of small pulse thresholds to achieve maximum overlap without introducing excessive transfer overheads. With eager DMA, cut-through delivery can significantly reduce latency for messages as small as a kilobyte on our platform.

Finally, we showed that the expected benefit of cut-through delivery remains significant even with faster I/O buses, e.g., high-quality 64-bit PCI I/O buses, assuming the bus can deliver high bandwidth for transfers in the 1K range. We believe that interconnect link speeds will continue to match I/O bus bandwidths, and that cut-through delivery will be recognized as a fundamental technique in the design of adapters and message systems when large-packet latency is important.

Acknowledgements

We thank Bob Felderman, Rich Martin, Scott Pakin, Mike Feeley, Chandu Thekkath, Jeff Mogul, Marc Viredaz, Geoff Voelker, and MSU for answering our questions about the Myrinet network, MCP design, GMS, AM and FM, and DEC PCI bridges and ATM products. Mike Feeley, Scott Pakin, Darrell Anderson, Geoff Cohen, Syam Gadde, and Carla Ellis commented on drafts of this paper.

References

[1] A. Basu, V. Buch, W. Vogels, and T. von Eicken. U-Net: A user-level network interface for parallel and distributed computing. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, 1995.

[2] M. A. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. W. Felten, and J. Sandberg. Virtual memory mapped network interface for the SHRIMP multicomputer. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 142–153, Apr. 1994.

[3] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W.-K. Su. Myrinet - a gigabit-per-second local area network. *IEEE Micro*, February 1995.

[4] G. Buzzard, D. Jacobson, M. Mackey, S. Marovich, and J. Wilkes. An implementation of the Hamlyn sender-managed interface architecture. In *Proceedings of the Second Symposium on Operating Systems Design and Implementation*. USENIX Association, October 1996.

[5] D. Comer and J. Griffioen. A new design for distributed systems: the remote memory model. In *Proceedings of the 1990 Summer USENIX*, June 1990.

[6] M. D. Dahlin, R. Y. Wang, and T. E. Anderson. Cooperative caching: Using remote client memory to improve file system performance. In *Proceedings of the First Symposium on Operating System Design and Implementation*, pages 267–280, November 1994.

[7] Z. D. Dittia, G. M. Parulkar, and J. R. Cox. The APIC approach to high performance network interface design: Protected DMA and other techniques. In *Proceedings of IEEE Infocom*, 1997. WUCS-96-12 technical report.

[8] P. Druschel, L. L. Peterson, and B. S. Davie. Experiences with a high-speed network adaptor: A software perspective. In *SIGCOMM '94*, pages 2–13, Aug. 1994.

[9] M. J. Feeley, W. E. Morgan, F. H. Pighin, A. R. Karlin, and H. M. Levy. Implementing global memory management in a workstation cluster. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, 1995.

[10] E. W. Felten and J. Zahorjan. Issues in the implementation of a remote memory paging system. Technical Report 91-03-09, Department of Computer Science and Engineering, University of Washington, March 1991.

[11] M. P. I. Forum. The MPI message passing interface standard. Technical Report 9, University of Tennessee, Knoxville, London, April 1994.

[12] H. A. Jamrozik, M. J. Feeley, G. M. Voelker, J. E. III, A. R. Karlin, H. M. Levy, and M. K. Vernon. Reducing network latency using subpages in a global memory environment. In *Proceedings of the Seventh Symposium on Architectural Support for Programming Languages and Operating Systems (ASPLOS VII)*, pages 258–267, October 1996.

[13] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks 3, pages 267-286, 1979.*, 3:267–286, 1979.

[14] R. Martin, L. T. Liu, V. Makhija, and D. Culler. LANai active messages (LAM). At URL http://now.cs.berkeley.edu/AM/lam_release.html.

[15] S. Pakin, V. Karamcheti, and A. Chien. Fast Messages (FM): Efficient, portable communication for workstation clusters and massively-parallel processors. *IEEE Parallel and Distributed Technology*, 1997.

[16] C. A. Thekkath and H. M. Levy. Limits to low-latency communication on high-speed networks. *ACM Transactions on Computer Systems*, 11(2), May 1993.

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