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.

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.

[*]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

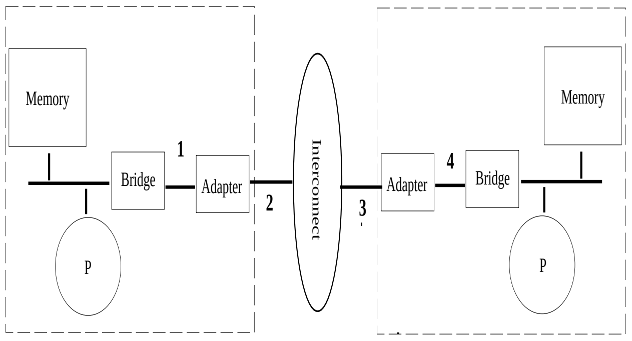

Figure 1. Four DMA Transfers for a Network Message

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.

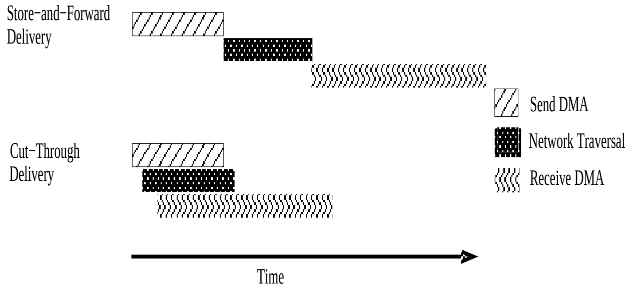

Figure 2. Cut-Through Delivery vs. Store-and-Forward on the AlphaStation 500

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,

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.