Design and Performance of Maestro Cluster Network

Shinichi Yamagiwa, Munehiro Fukuda and Koichi Wada
Institute of Information Sciences and Electronics
University of Tsukuba,
Tsukuba, Ibaraki 305-8573, JAPAN
e-mail: {sy, fukuda, wada}@is.tsukuba.ac.jp

Abstract

Most clusters so far have used WAN or LAN-based network products for communication due to their market availability. However, they do not always match communication patterns in clusters, thus incurring extra overhead. Based on our investigation for such overhead, we have optimized cluster communication at link layer. Partitioning each message in 16-byte packets, our optimization uses two techniques: (1) transferring in burst as many packets as the receiving buffer accepts at once, and (2) having each hardware component pass one packet to another in a pipelined manner. We have realized those two techniques in a link control hardware chip, referred to as MLC(Maestro Link Controller), and have constructed the Maestro cluster network using MLCs. This paper describes the feature of the Maestro cluster network and demonstrates the efficiency of our optimization techniques through performance experiments over this network.

1 Introduction

The emergence of high-performance microprocessor has made it attractive to use a PC cluster as a parallel computing system with an excellent cost/performance ratio [12, 13]. Most PC clusters so far have used WAN or LAN-based network devices and protocols such as Ethernet and TCP/IP, due to their market availability. However, they do not always match communication patterns in clusters[5], thus deteriorating the entire performance with extra overhead incurred from each layer of (1) communication software, (2) device handlers and (3) network hardware.

Performance improvements in communication software have been well studied in FM[15], PM[3], and BIP[6]. Therefore, we have aimed at network hardware. Partitioning each message in 16-byte small packets, we have applied two optimization techniques to link-layer communication: (1) transferring in burst as many packets as the receiving buffer accepts at once, and (2) having each hardware component pass one packet to another in a pipelined manner. The former has potential to reduce network latency, and the latter to improve network throughput.

Based on those two optimization techniques, we have designed a novel link protocol for PC clusters, referred to as *Maestro Link Protocol* [16], have implemented the protocol in a hardware chip, termed *MLC(Maestro Link Controller)*, have constructed the Maestro cluster network using MLCs, and have conducted the performance evaluation. The results demonstrate the efficiency of the proposed techniques for cluster communication, (i.e. both low network latency and high throughput).

The remainder of our paper is as follows: Section 2 investigates communication overheads in PC clusters. Section 3 presents Maestro Link Protocol and describes the Maestro cluster network. Section 4 discusses the performance. Section 5 compares our network with others, and Section 6 concludes the discussions.

2 Conventional Approaches

Of importance is how to support two different aspects of communication: low latency and high throughput. The former must be emphasized on for frequent inter-processors synchronization as seen in distributed simulations, and the latter for mass data transfer as in matrix computations. In the following, we will discuss about the main obstacles to low latency and high throughput in (1) communication protocol software, (2) device handlers, (3) data link layer, and (4) physical layer.

1. *Communication Protocol Software*

   The problem is the repetition of data-copying operations before messages have been transferred and made available to their receiving applications. In general, each message must be copied between the user and the kernel space, and may be further copied between the kernel space and the network device memory. Nevertheless, such copying operations increase communication latency. In addition, the user-to-kernel context switches can be considered as another overhead, especially in case of frequent inter-processors synchronization.

This problem has been well addressed by FM[15], PM[3], and BIP[6] communication libraries. They have realized the zero-copy communication that permits underlying network devices to access messages directly in the user space, thus bypassing the kernel intervention.

2. *Device Handlers*

   The virtual-to-physical address translation is an obstacle to bypassing the kernel intervention. Since communication buffers at user level are allocated in the virtual address space, their physical addresses must be identified prior to the actual transfers by underlying network devices. If those devices maintain the page table and TLB, (i.e. Translation Look-aside Buffer), they can take charge of such translation and therefore transfer user messages without the kernel support. This also gives more CPU time for user applications and hides communication overhead further.

3. *Data Link Layer*

   The first problem is conventional message framing. If the minimal frame size is too large, network latency increases relatively in transferring shorter messages. On the other hand, if the maximal frame size is too small, a longer message must be sent with multiple frames, which increase framing operation and thus degrade the total throughput. The Ethernet frame for instance must include 36-byte link-by-link routing information, which is in turn the minimal frame size [14]. The maximal frame size is also restricted up to 1500 bytes, so as not to monopolize slow communication link. However, assuming that PC clusters are used in a geographically closed environment, messages can be sent directly to their destination with less routing information. Hence, we must design a new message frame suitable to cluster communication rather than use such the redundant Ethernet frame.

   The second problem is excessive DMA invocations. Most network devices always invoke DMA transfers to send data over network, whether or not the data size is too small to use DMA efficiently. This is regarded as overhead especially when fine-grained communication is repeated for frequent inter-processors synchronization. Therefore, network devices should include another high-speed transfer logic for a small amount of data, and switch between this logic and DMA according to the data size.

   The third problem is the current transfer scheme to transmit data on physical link. Provided a packet be a transfer unit dealt at physical layer, most network devices do not aggregate independent packets in one, whether or not those packets are too small. In other words, they repeat an entire transfer set-up operations including physical link arbitration for each small packet. This makes physical link idle frequently, degrading network throughput consequently. To address this problem, network devices must be capable of packet aggregation.

4. *Physical Link Layer*

   Multicast takes an important role in the master-slave or client-server model. Conventional multicast either broadcasts a message to all PCs and has non-receivers discard it, or sends a copy of message to each receiver one after another. The former is familiar to carrier-sensitive media such as Ethernet, however its drawback is unnecessary message elimination at non-receivers. The latter is easy to implement, while incurring excessive message-copying operations. Thus, the physical link layer must facilitate a multicast scheme that involves neither discarding nor copying messages.

Among those four layers, most research work in cluster computing has been focusing on the communication protocol software. In terms of device handlers, Myrinet[9] and U-net[18] have realized the virtual-to-physical address translation in their network devices. Therefore, our main focus is performance improvements in the lower two layers such as data link and physical link layers, discussed in the following sections.

3 Maestro Cluster Network

3.1 Key Techniques

To address the performance drawbacks at data link layer, we propose two optimization techniques below:

1. *Packet Aggregation*

   Partitioning each message in fine-grained packets, this technique aggregates multiple packets best fit to the space currently made available to the receiver's buffer, and transfers them in burst to the receiver. The finer packet size the better buffer utilization. This packet aggregation does not matter the message boundary. This means that one aggregate may include multiple messages or message chunks over their boundary. The

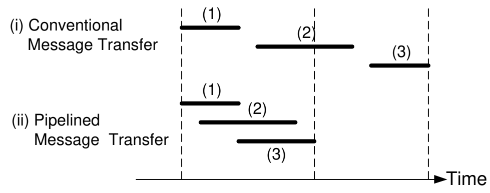


Figure 1. Conventional vs. pipelined transfer

burst transfer of aggregated packets reduces the number of physical link arbitrations, each followed by miscellaneous device set-up operations. Therefore, we expect that this technique improves network throughput.

In the following discussion, we refer to this transfer scheme as *Network Burst*. In addition, we simply define each fine-grained packet as a *packet* and aggregated packets as a *frame*.

2. *Pipelined Transfer*

Conventional network devices do not simultaneously take care of two or more messages. In other words, once such a device accepts a new message from its host, the following messages are totally blocked until the device completes transmitting the current message onto physical link. This causes the serialization of the following three stages of message transfer: (1) the one from the host memory to physical link at a sender side, (2) the one through physical link, and (3) the one from the physical link to the host memory at a receiver side. Figure 1-(i) depicts this serialization. (In this figure, note that the number associated with each horizontal line corresponds to the stage order.)

To avoid this serialization, we again partition each message into fine-grained packets, and have each device component pass one packet to another in a pipelined manner. As soon as the sender network device transmits a packet onto physical link, it picks up the following packets from the host memory. Similarly, the receiver device keeps storing packets to the host memory as receiving the following packets from physical link. Therefore, the three serialized stages are simultaneously processed in a pipelined fashion. The Figure 1-(ii) illustrates how network latency is shrunk by the pipelined transfer. The finer packet size and more device components, the more pipelined messages. This means that the technique also improves transfer throughput.

3.2 Maestro Link Protocol and Controller

Based on our optimization techniques, we have designed a novel link protocol, called *Maestro Link Protocol*. As shown in Figure 2, the protocol assumes the point-to-point half-duplex link such as IEEE1394 [4] physical layer(IEEE1394PHY). Its host interface is realized with a pair of sending and receiving FIFO buffers that exchange messages with the host machine.

While the sender host keeps passing messages to the sending FIFO, those messages are pipelined and partitioned one after another into finer-grained packets, which are then aggregated into as a large frame as the receiving FIFO can store, and are thereafter transferred on physical link in *Network Burst*. On the receiver side, those packets are reassembled into the original messages that are finally made readable from the receiving FIFO to the host. Many-to-many host communication needs the switch that establishes a Maestro-Link-Protocol connection with each host and routes a message from the source to the final destination host.

To facilitate both *Network Burst* and pipelined packet transfer, the protocol provides the following three features:

1. *FIFO flow control*

   The protocol defines its status register that indicates how many bytes the host can write to and read from its FIFO buffers, rather than if the host can send or receive a message. This scheme maximizes the message-spooling capability in the actual implementation, prompts the host to exchange the next available message quicker, and therefore mitigates network latency. This advantage is further exploited by having two or more pairs of FIFO buffers in the actual implementation (in which case we refer to each pair as a *channel*.)

2. *Fair link arbitration*

   The fair link arbitration is the key to realize pseudo-full duplex transfer between two end points of a half duplex link. Otherwise, one end point may eagerly use physical link and starve out the other end that consequently suffers from higher network latency. To avoid this situation, Maestro Link Protocol requests the current sender to release physical link every *Network Burst* transfer, and immediately grants the receiver to use the link. However, this rule does not work out unless both ends continue performing a transfer by turns. To maintain the mutual grant to the both ends, the protocol requests the current receiver not to turn down the grant, whether or not it has packets to send. If the receiver has no packets, it must simply send a null frame, (i.e. the one that does not include any packets) so as to invoke the next link arbitration. Hence,

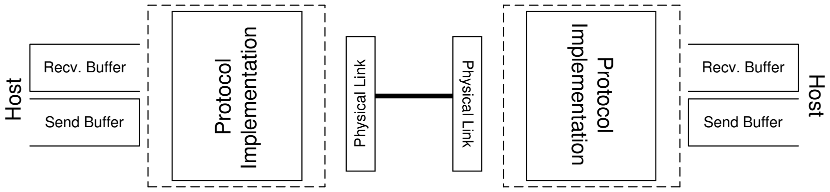


Figure 2. Maestro Link Protocol

Table 1. Protocol specification

Spec. items	Description	
Physical Link	Layer standard	IEEE1394
	Transfer speed	200 Mbps
	Data width	4 bits
Frame	♯ packets	Up to 15 packets
	Packet size	16 bytes
FIFO buffer	♯ channels	Two (Ch0 and Ch1)
	Host-side data width	16 bits
	Capacity	2 Kbytes

the both ends keep granted to use half-duplex link by turns.

3. *Physical link flow control*

   The *Network Burst* transfer needs to know *a priori* the FIFO capacity currently available at the receiver side. This requirement is satisfied with carrying such capacity information in each frame. This is because the fair link arbitration guarantees two end points to exchange a frame with each other alternately. We call this capacity information a *credit*, defined as the number of transferable packets.

We have realized these features in an *Altera MAX7256-7* FPGA chip [2], with the protocol specification as shown in Table 1. The chip includes 5000 gates and functions at 50 MHz. We call this chip *MLC (Maestro Link Controller.)*

Of great concern is the packet size. As mentioned before, the smaller packet size the better pipelined communication. Should a packet size be however too small, we would have to prepare hardware logic large enough to count a huge number of packets stored in each FIFO. To implement MLC in a reasonable amount of logic, we have defined a packet in 16 bytes.

MLC provides the host with the following interface: The host must first compose each message of packets as specified in Figure 3, before pushing it to a sending FIFO. The message must contain at least one packet whose first five bytes are used to store the routing information rather than user data, which must be thus placed in the last 11 bytes. When the message consists of two or more packets, the second packet must use the first two bytes, (referred to as *Pkt-*

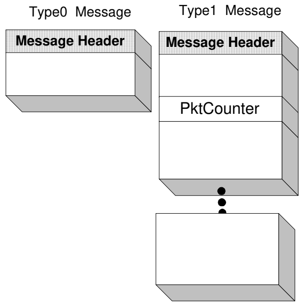


Figure 3. Message format

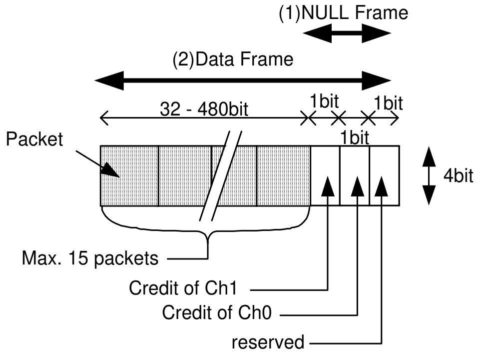


Figure 4. Frame format

Counter) to indicate the number of packets totally contained in the message. A message with a single packet and the one with multiple packets are distinguished as Type0 and Type1 respectively.

The frame transfer to IEEE1394PHY is achieved as follows: After pulled out from a sending FIFO, packets form a frame as defined in Figure 4. Since IEEE1394PHY is four bits wide, each 16-byte packet is formatted into a 4×32-bit sequence. The frame begins with a 4×3-bit preamble including the Ch0 and Ch1 credits and followed by up-to-15 packets, (i.e. 240-byte user data at maximum.) This means that the frame size varies from 4×3 bits to 4×483 bits. The frame with only a preamble is the null message used to prompt the next fair link arbitration, and is distinguished from the data frame that contains packets.

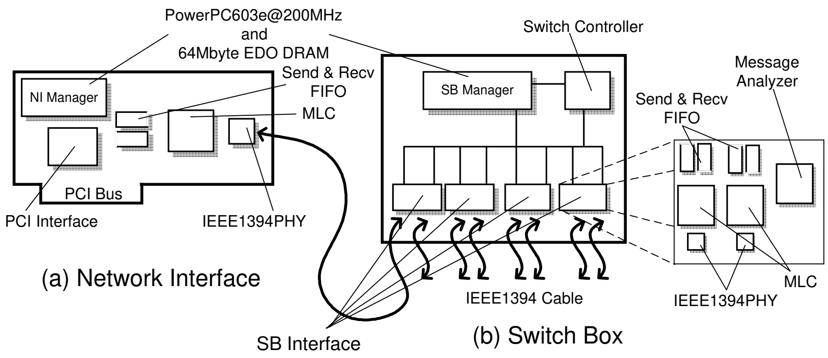


Figure 5. Maestro cluster network

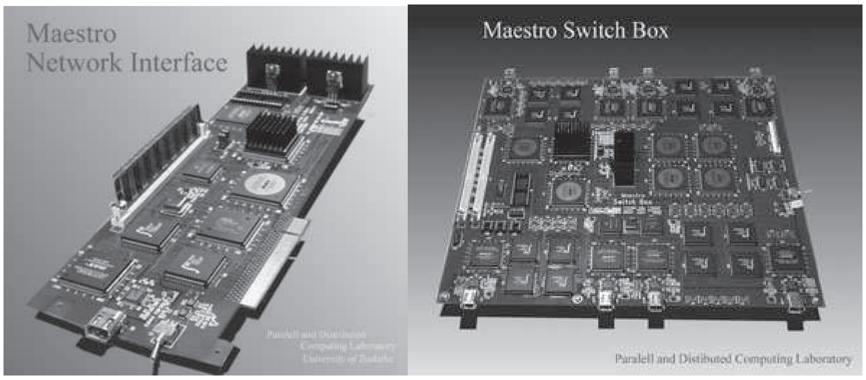


Figure 6. Maestro network interface and switch box

3.3 Cluster Implementation

To assess the correctness and performance of Maestro Link Protocol, we have implemented the Maestro cluster network where cluster nodes communicate with each other via MLCs.

As illustrated in Figure 5, the Maestro cluster network is composed of network interface cards, each adapted to a PC via its PCI bus [10], IEEE1394 cables, each emanating from a network interface card, and a central switch box that terminates all the cables to it and that routes messages from the source to destination network interface cards. In following discussion, we abbreviate the network interface card and the switch box to NI and SB respectively.

1. NI

   NI consists of the following five hardware components: a PCI interface, an NI manager, two FIFO buffers, an MLC and IEEE1394PHY.

   With its DMA controller, the PCI interface [11] takes charge of message transfers from the host memory not only to the NI manager's DRAM but also to the FIFO buffers. It also permits the host to get direct access to the DRAM and FIFO buffers. The NI manager is comprised of PowerPC603e@200MHz and 64-Mbyte EDO DRAM, handling the zero-copy communication as well as the virtual-to-physical message address translation. It also takes charge of host-to-NI message transfer in case when the message is too small to amortize DMA set-up overhead.

   The rest of the components, (i.e. the FIFO buffers, MLC, and IEEE1394PHY) have the specifications and behaviors described in Section 3.2.

2. SB

   SB consists of the following six hardware components: four SB interfaces, an SB manager, and a switch controller.

   The SB interface includes two sets of IEEE1394PHY, MLC, and two FIFO buffers. Each set exchanges messages with a different host's NI. (Note that all those components are the same as NI's.) The SB interface invokes its internal logic, called *message analyzer*, to find the type of a newly incoming message, (i.e. Type0 or Type1 as shown in Figure 3), to extract its message header, and to pass it to the SB manager.

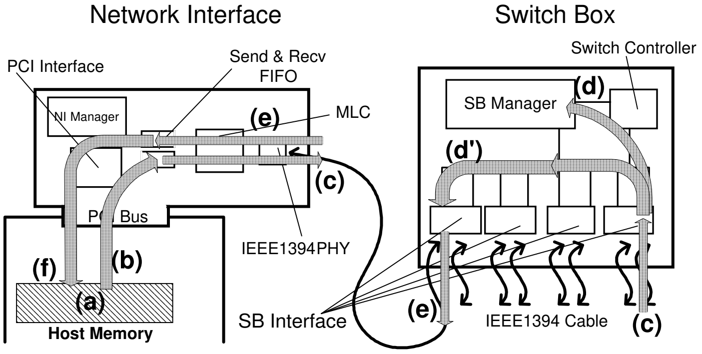


Figure 7. Message flow

The SB manager consists of a PowerPC603e @200MHz and 64-Mbyte EDO DRAM. It accepts a new message header from each message analyzer of the four SB interfaces in a round-robin fashion. It analyzes this header to decide the message destination, and requests the switch controller to route the message from the source to the destination SB interface's FIFO buffer.

The switch controller uses its own DMA controller to transfer each message from one FIFO buffer to another at the rate of 400 Mbps. As shown in Figure 5, all the FIFO buffers are connected to the shared bus, which permits the switch controller to multicast a message to two or more FIFO buffers at once. This multicast mechanism eliminates the message-copying overhead discussed in Section 2. The switch controller also provides all cluster nodes with fast barrier synchronization by counting a barrier request from each node and broadcasting a barrier acknowledgment on the shared bus which is thus delivered to each node.

Figure 6 shows our first implementation of NI and SB. We have confirmed that the Maestro cluster network is functioning as its specification for an eight-PC cluster. The network scalability can be addressed by constructing a hierarchical tree network where the leaves are PCs and the other nodes are SBs. Since each node of this tree network has a pair of receiving and sending FIFO buffers, the dead lock will never occur.

3.4 Comprehensive Message Flow

Using Figure 7, we explain a comprehensive message flow from one host memory to another.

Assume that the OS allocates a certain amount of space as a user-level communication buffer upon a system initialization [1]. We call this space *reserved memory*. The device handler takes care of virtual-to-physical address translation for this reserved memory. Given a 300-byte message in the reserved memory (Figure 7-(a)), the sender host then transfers it to the sending FIFO buffer using the PCI interface's DMA controller or the NI manager's PowerPC (Figure 7-(b)). Upon receiving the first 16-byte data, the MLC starts formatting it into frames (Figure 7-(c)). Since each frame can contain 240 bytes at maximum, the message is formatted into two frames and consequently sent to SB in two *Network Burst* transfers. Between these two transfers, SB is given an opportunity of transferring back any (different) message to the sender host, which in turn realizes well-balanced bi-directional communication.

Receiving the header of this 300-byte message (Figure 7-(d)), SB manager decides the message destination and requests the switch controller to transfer the entire message from the source to the destination FIFO buffer with its DMA controller(Figure 7(d')). Note that this DMA transfer is invoked only once even in case when having two or more destinations. This is because the corresponding FIFO buffers receive the message on the shared bus simultaneously.

Thereafter, the message is again framed and transferred in burst to the receiver NI (Figure 7-(e)), where the original message is reconstructed and restored in the host's reserved memory. The receiver NI performs this message restoration in the reversed order of the message packetization at the sender NI (Figure 7-(f).) Again note that the receiver MLC overlaps *Network Burst* transfers with the message restoration onto the reserved memory once it receives the first 16-byte packet.

In summary, this message-flow example demonstrates that the Maestro cluster network has addressed all the problems in cluster communication as described in Section 2. Using the same message flow, we have empirically evaluated its performance in the following section.

4 Performance Evaluation

This section demonstrates the performance of the Maestro cluster network from the following three points of view:

(1) the basic hardware performance, (2) the effect of *Network Burst*, and (3) the effect of pipelined transfer. For all the experiments, we have used the same message flow discussed in Section 3.4. Network latency was measured with the time base register of NI's PowerPC. The measurement starts when the sender host invokes its NI's DMA controller to transfer a message from the host memory to the NI, and it ends when the sender receives a Type0 message of 16 bytes as an acknowledgement from the receiver. In the following figures, we simply refer to this round-trip network latency as a *remote-DMA latency*.

4.1 Basic Hardware Performance

1. *Inter-MLCs Communication Performance*

   The inter-MLCs communication performance was measured using a 100 MHz-sampling logic analyzer. Figure 8 shows its timing chart when a 16-byte Type0 message was transferred from one MLC to another via SB. Note that t_1 and t_4 in Figure 8 includes the maximal and the minimal link-arbitration time, respectively. The sender MLC needs t_1 when the host writes a new message to the sending FIFO immediately after the MLC releases physical link. Even taking account of the difference between t_1 and t_4, (i.e. 1.12 μsec), the maximal latency of inter-MLCs communication costs approximately 8 μsecs at most. It is more than 15 times smaller than the IEEE1394 general link layer's maximal latency, (i.e. 125 μsec).

   Figure 9 shows the throughput of PingPong communication over the Maestro cluster network. The maximal throughput is 20 Mbytes/sec, (i.e. 160 Mbps). This demonstrates that the network achieves up to 80% of 200-Mbps IEEE1394PHY's peak performance

   We have also measured the throughput of PingPing communication, where two NIs send their messages to each other via SB at a time. Specifically to say, upon a synchronization with SB, two NIs invoke their DMA to send a message to and thereafter receive an acknowledgment from each other. Out of two performance results measured at each NI respectively, we took the worse one as the PingPing performance. The result shows that PingPing communication achieved 10-Mbyte/sec throughput, equal to 50% of the PingPong performance.

2. *Switching Performance*

   SB's message switching latency is regarded as the sum of SB's message analysis time and its inter-FIFOs message transfer time, referred to as t_2 and t_3 in Figure 8, respectively. The t_2 measurement was started when the SB manager reads a message header from a message analyzer's FIFO buffer, and was completed when it requests the switch controller to route the message. The t_3 measurement was initiated when the switch controller invokes DMA to send a message from one FIFO to another inside SB, and was completed when the DMA finishes the message transfer. The DMA setup needs only 40 nsec. The DMA achieves 400-Mbps message transfer when sending a Type0 message. This transfer rate is 2.5 times higher than inter-MLCs communication throughput. Therefore, SB does not slow down duplex communication.

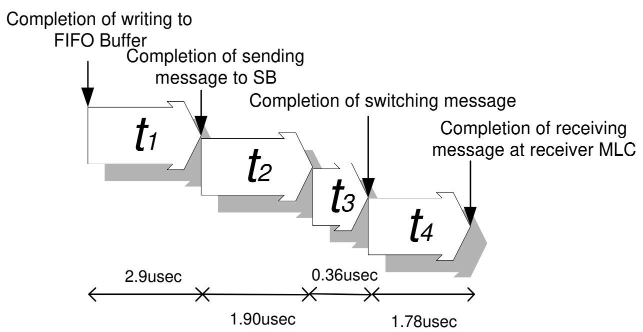


Figure 8. Inter-MLCs communication performance

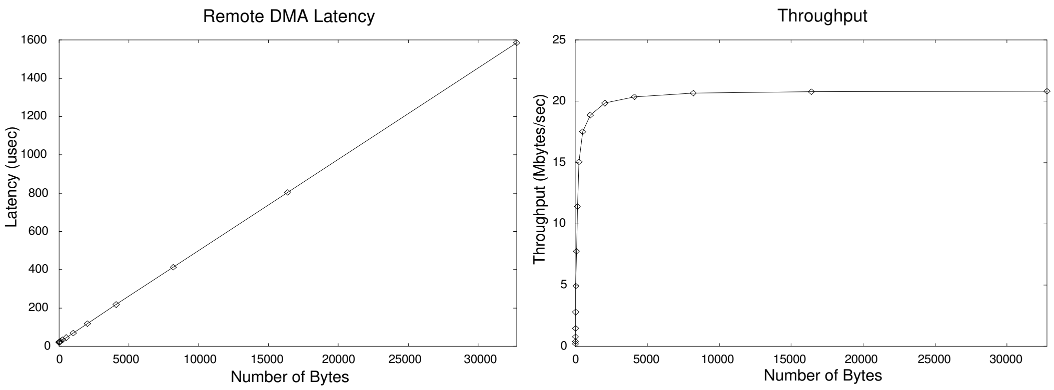
4.2 Effects of Network Burst

Figure 10 shows the performance comparison between *Network Burst* and conventional single packet transfers. The former aggregates as many 16-byte packets as possible to transfer at a time. The latter transfer fixes the packet size to 16, 32, and 64 bytes. The larger packet size, the higher throughput for single packet transfers. For instance, a 64-byte packet is transferred at 1.6 times higher throughput than a 16-byte packet, while its minimal latency is three times larger. In any cases, *Network Burst* achieves better than single packet transfers.

4.3 Effects of Pipelined Transfer

Figure 11 shows the comparison between our pipelined transfer and conventional message-based transfers.

To realize conventional message-based transfers with MLCs, we need to inactivate its feature of pipelined transfer. In this inactive mode, the sender MLC delays transferring a message onto physical link until the entire message has been stored in the sending FIFO, as well as the receiver MLC delays writing the message to the host memory until the entire message has been buffered in the receiving FIFO. We conducted the experiment as changing the message size



Number of Bytes	16	32	64	128	256	512	1024	2048
Latency (μsec)	21	22	23	26	33	45	68	117

Figure 9. Maestro Cluster Network's basic performance

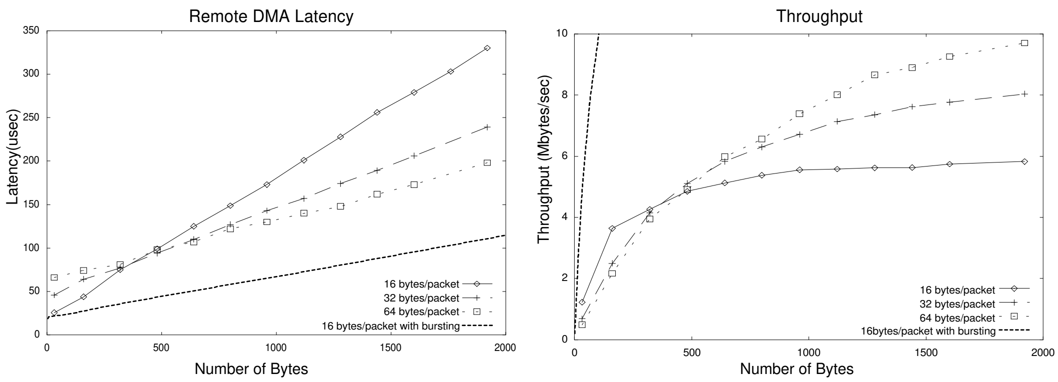
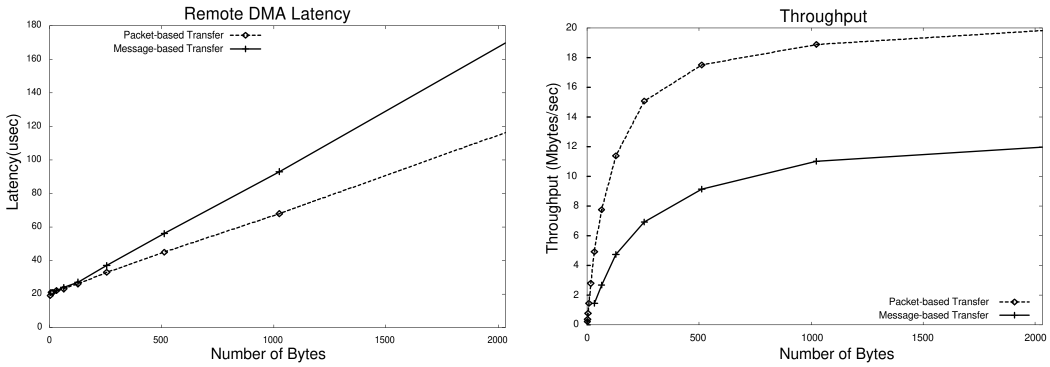


Figure 10. Network Burst vs. single packet transfers



Number of Bytes	16	32	64	128	256	512	1024	2048
Latency of Packet-based Transfer (μsec)	21	22	23	26	33	45	68	117
Latency of Message-based Transfer (μsec)	21	22	24	27	37	56	93	171

Figure 11. Pipelined vs. message-based transfers

from 16 bytes through to 2 Kbytes, (i.e. the maximal FIFO buffer size).

In our pipelined transfer, the sender MLC starts sending the message header to physical link while still receiving the entire message from the sender host memory, because of which the receiver MLC can immediately invoke a DMA message transfer to its host memory. Hence, both the sender and receiver MLCs can work on message transfer in parallel. For transferring a 16-byte message, the results show little difference in performance between our pipelined transfer and the conventional message-based transfer. However, our pipelined transfer demonstrates lower latency and higher throughput than the conventional transfer as increasing the message size. From the results, we have confirmed that the pipelined transfer performs at maximum 8 Mbytes/sec faster than the conventional transfer.

5 Related Work

In this section, we will differentiate our optimization techniques from other work in terms of *Network Burst*, pipelined transfer, and performance.

1. *Network Burst*

   TCP's sliding window technique improves network throughput by sending message frames asynchronously. However, as passed to IP and Ethernet device handlers, each frame adds to itself an excessive amount of header and footer information, which slows down the communication speed.

   Myrinet uses one bit of its physical link, termed a *B* bit, to perform burst message transfer under its *STOP-and-GO* flow control. It keeps sending data on its eight-bit-wide physical link till receiving a *STOP* signal. However, such implementation needs more *B* bits as preparing multiple communication buffers.

   Maestro Link Protocol on the other hand efficiently aggregates packets to send at a time, needs no flow-control bit like Myrinet's *B* bit, and thus allows two or more channels of FIFOs to perform *Network Burst* concurrently without extending the link width.

2. *Pipelined Transfer*

   TCP's fragmentation is similar to our optimization in dividing a user message into smaller pieces. However, the difference is that TCP treats such pieces as independent messages, each containing full header and footer information and thus being routed alone. Maestro's packets on the other hand form an aggregation as a routing unit, and therefore, each packet need not include full routing information. Furthermore, in TCP, the host machine performs message fragmentation as triggered by I/O interrupts, which thus incur more communication overhead.

   FM2.0 with Myrinet [7] realizes pipelined message transfer by providing a user with library functions that divide a message into packets and transfer each packet one by one. However, Myrinet needs to temporarily store packets in its SRAM. Due to this reason, no matter how small packet the host sends actually, it must invoke an expensive DMA transfer to Myrinet. In addition, Myrinet terminates an SRAM-to-physical-link DMA transfer whenever encountering the delimiter of the current message in transfer. This means that pipelined message transfer is always cut off over each message boundary. Our pipelined transfer on the other hand does not matter such message boundary. Packets in different messages can form as a large aggregation as the receiver can accept. Hence, we can reduce idle time in communication and realize seamless pipelined message transfers.

3. *Performance*

   Table 2 compares the Maestro cluster network with Gigabit Ethernet [17] and Myrinet [8] in terms of latency at link layer from the sender network device to the switch, inside the switch, and from the switch to the receiver network device, each denoted as NI-SW, SW, and SW-NI, respectively. Due to their different architecture, their smallest message size varies from 14 to 16 bytes. The table shows that Maestro works 1.5 times better than Gigabit Ethernet, while performing twice slower than Myrinet. This is because Myrinet uses a communication-specific processor, named *LANai*.

   We have also compared their network throughputs for a 1500-byte *remote-DMA* transfer. Based on [17], we have calculated that Gigabit Ethernet's throughput is 313 Mbps, (i.e. 31% of its peak performance.) Our experiment using Myrinet has resulted in 113 Mbps. This is 18% of its one-way peak performance, (i.e. 640 Mbps.) Maestro's throughput is, on the other hand, 68% of its peak performance.

In summary, while TCP, Myrinet, and FM2.0 have employed similar techniques to ours, we have much more focused on the performance improvement at link and physical layers than those systems.

6 Conclusions

Based on the overhead investigation for cluster communication, we have proposed two optimization techniques at link layer: packet aggregation and pipelined transfer. Maestro Link Protocol is the one that has incorporated these

Table 2. Latency comparison

Networks (message size)	NICs, Switches (Manufacturer)	NI-SW (μsec)	SW (μsec)	SW-NI(μsec)	Total (μsec)
Maestro (16 bytes)	Maestro NI, Maestro SB	1.78	2.26	1.78	5.82
Gigabit Ethernet (14 bytes)	EC-440-SF,Summit2 (Extreme Networks)	1.30	5.00	2.50	8.70
Myrinet (15 bytes)	PCI32(LANai 4.3, 512Kbyte SRAM), M2M-DUAL-SW8 (Myricom)	0.67	1.40	0.67	2.74

two optimizations as well as the idea of fair link arbitration. We have realized this protocol in a hardware chip, (i.e. MLC), have implemented the Maestro cluster network where nodes communicate with each other via their MLCs and a switch box, and have evaluated the performance. The results demonstrated that our optimization techniques contribute to the performance improvement for cluster communication.

Maestro Link Protocol is applicable to any point-to-point communication medium such as USB (Universal Serial Bus) [19]. Using the next generation of IEEE1394PHY capable of 400-Mbps or 800-Mbps transfer, MLC would achieve 320-Mbps or 640-Mbps throughput respectively. Although our current implementation of *credit* is denoted in 1×4 bits and thus limited to specifying up to 15 packets, a new eight-bit-wide IEEE1394PHY will extend this *credit* size, which permits more packets to be aggregated and further improves the throughput.

Nevertheless, MLC needs to be modified as IEEE1394PHY is revised for higher-speed communication. The modification is however only expanding MLC's data bus from 4-bit to 16-bit width, which can remove its internal 4-to-16 data steering logic and thus simplify MLC's entire logic.

Our future plans include (1) implementing a user-level communication library to maximize the benefits from the Maestro cluster network, (2) implementing a distributed shared memory environment over this network, (3) applying Maestro Link Protocol to the next generation of IEEE1394 physical layer and optical communication media, and (4) conducting further performance evaluation.

References

[1] Alessandro Rubini and Andy Oram. *Linux Device Drivers, Chapter 13*. O'Reilly & Associates, 1998.

[2] Altera Corporation. *1998 Data Book*. 1998.

[3] Hiroshi Tezuka et al. PM: An operating system coordinated high performance communication library. In *High-Performance Computing and Networking, volume 1225 of Lecture Notes in Computer Science*, pages 708–717. Springer-Verlag, 1997.

[4] IEEE Standard Department. *IEEE Standard for a High Performance Serial Bus*. 1994. http://www.1394ta.org.

[5] V. Karamcheti and A. Chien. Software overhead in messaging layers: Where does the time go? In *Proceedings of International Conference on Architectural Support of Programming Languages and Operating Systems (ASPLOS-VI)*, pages 526–531, 1994.

[6] Loic Prylli and Bernard Tourancheau. BIP: a new protocol designed for high performance networking on myrinet. In *Workshop PC-NOW, IPPS/SPDP98*, Orlando, USA, 1998. Elsevier Science Publishers.

[7] Mario Lauria et al. Efficient layering for high speed communication: Fast messages 2.x. In *Proceedings of the 7th High Performance Distributed Computing (HPDC7) conference*, Chicago, Illinois, 1998.

[8] Motohiko Matsuda et al. Network interface active messages on SMP clusters. *Technical Report on IPSJ SIGARC*, Vol.97(No.125):55–60, 1997.

[9] Nannette J. Boden et al. Myrinet - a gigabit-per-second local-area network. *IEEE Micro*, Vol.15(No.1), 1995.

[10] PCI Special Interest Group. *PCI Local Bus Specification. Rev. 2.1*. 1995.

[11] PLX Technology. *PCI9060 Data Sheet VERSION1.2*. 1995.

[12] Rajkumar Buyya. *High Performance Cluster Computing: Architectures and Systems*. Prentice Hall, 1999.

[13] Rajkumar Buyya. *High Performance Cluster Computing: Programming and Applications*. Prentice Hall, 1999.

[14] Robert Breyer and Sean Riley. *Switched and Fast Ethernet, Second Edition*. Ziff Davis Press, 1996.

[15] Scott Pakin et al. Fast messages (FM): Efficient, portable communication for workstation clusters and massively-parallel processors. *IEEE Concurrency*, Vol.5(No.2):60–73, 1997.

[16] Shinichi Yamagiwa et al. Network Architecture and Communication Protocol for Cluster Computers – Developmaent and Performance Evaluation of Maestro Network. *IPSJ Transactions on High Performance Computing Systems*, Vol.41(No. SIG 5(HPS 1)), August 2000.

[17] Sinji Sumiyoshi et al. The design and evaluation of high performance communication library using a gigabit ethernet. *Technical Report on IPSJ SIGHPC*, Vol.72(No.19), 1998.

[18] T. von Eicken et al. U-Net: A user level network interface for parallel and distributed computing. In *In Fifteenth ACM Symposium on Operating Systems Principles*, pages 40–53, 1995.

[19] USB Implementers Forum Inc. *Universal Serial Bus Revision 1.1 Specification*. http://www.usb.org/.