Firmware Support for Reliable Communication and Dynamic System Configuration in System Area Networks

Princeton University Technical Report TR-581-98

Angelos Bilas, Cezary Dubnicki, Stefanos Damianakis, Yuqun Chen, and Kai Li
Computer Science Department, Princeton University, Princeton, NJ-08544
{bilas, dubnicki, snd, yuqun, li}@cs.princeton.edu

Abstract

Research efforts to provide fast multicomputer interconnection networks for clusters of workstations (SANs) at low cost has lead to the design of network interfaces that support minimal standard functionality in the network interface and use powerful basic blocks that allow communication system designers to implement the functionality needed by different paradigms and systems using these basic blocks. These network interfaces usually encompass a general purpose processor, use simple switches to route packets to their destination, and allow for arbitrary network topologies. These characteristics result in the absence of reliability in the network interface hardware and the need to support routing for arbitrary, dynamic topologies in the communication system.

There are many ways to deal with these problems. However, if performance is not to be compromised, these problems are highly non-trivial and many issues need to be considered. Data copies need to be avoided, algorithms and protocols need to be simple and memory requirements must be low, since the network interface has limitations in both network processor speed and memory.

In this work we address these two issues by presenting the extensions we did to our communication system, called Virtual Memory Mapped Communication (*VMMC*), to support reliable communication in the network interface and dynamic system configuration. We propose and implement reliable communication in the firmware that runs on the network interface processor with minimal cost; less than 2.5μs for latency and less than 10% for all different types of bandwidth Moreover, we propose a new scheme for dynamically determining the network topology. This mechanism relies on the retransmission mechanism that is used to provide reliable communication. Since the support for dynamic system configuration is not on the common path, there is no effect on system performance when no changes in the system topology occur.

1 Introduction

The desire to build high-performance servers from a network of commodity computers has pushed researchers to explore ways to reduce the communication bottleneck for system area networks. During the last few years, researchers have proposed several approaches to move network protocols partially or entirely to the user level with protection in order to reduce the protocol overhead [1, 6, 7]. The common framework is to implement a user-level data transport protocol that integrates well with the network interface. Existing high-level protocols for shared virtual memory and message-passing can then be implemented efficiently. The goal is to deliver to the user communication performance close to the hardware's limit while providing the full functionality of the high-level APIs.

A challenging issue in the design and implementation of user–level communication systems is where and how to incorporate a retransmission protocol in order to provide low-overhead, reliable communication for system area networks. Traditionally, reliable end-to-end connections have been implemented in the TCP layer with retransmission. This is a satisfactory design for wide area networks but the overhead of TCP is significant, in particular, when it is implemented on the host computer. This is because a retransmission protocol requires a sender process not to touch its send buffer until the message is acknowledged. The sender has to either wait for the acknowledgment, copy the send buffer to a system buffer, or copy-on-write for the send buffer. Our approach is to implement the standard retransmission method at the data link level, that is, between network interfaces. This approach takes advantage of the buffering of outgoing packets on the network interface and eliminates the need for the sender to wait for acknowledgments or to copy send buffers. We show that the loss of bandwidth of this method is small and its impact on latency is only a few microseconds.

Moreover, the nature of these new programmable network interfaces is such that the burden of determining the routes to each host in the network and actually routing the packets to their destinations is left to the communication libraries. Most SAN networks use switches and source routing to connect multiple nodes in a network and to deliver packets to their destinations. We present a new method for determining the topology of the network and the routes to each node in the system. Our method determines the network topology dynamically on demand and supports arbitrary changes in the network configuration.

To demonstrate these techniques and their integration, we have designed an extended virtual memory-mapped communication model *VMMC-2* and implemented it on PCs connected with Myrinet (described later). We also implemented the stream sockets protocol on top of *VMMC-2* to demonstrate how to take advantage of the new features. Our microbenchmarks show that the one-way latency of *VMMC-2* is 13.4 μs with retransmission and dynamic mapping and the maximum bandwidth is over 90 MBytes/s. The one-way latency of the stream sockets layer is 20 μs and its bandwidth is over 84 MBytes/s. We experimented with several applications using the stream sockets facility and showed that these techniques are indeed effective.

2 Myrinet

Myrinet is a high-speed local-area network or system-area network for computer systems. A Myrinet network is composed of point-to-point links that connects hosts and switches. The network link can deliver 1.28 Gbits/sec bandwidth in each direction [2].

The Myrinet network provides in-order network delivery with very low bit error rates (below 10^{-15}). On sending, the 8-bit CRC is computed by hardware and is appended to the packet. On a packet arrival, CRC hardware computes the CRC of the incoming packet and compares it with the received CRC. If they do not match, a CRC error is reported.

We designed and implemented a VMMC mechanism for the Myrinet PCI network interface. The PCI network interface is composed of a 32-bit control processor called LANai (version 4.1) with 256 KBytes of SRAM (Static Random Access Memory). The SRAM serves as the network buffer memory and also as the code and data memory of the LANai processor. There are three DMA engines on the network interface: two for data transfers between the network and SRAM and one for moving data between the SRAM and the host main memory over the PCI bus. The LANai processor is clocked at 33 MHz and executes a LANai control program (LCP) which supervises the operation of the DMA engines and implements a low-level communication protocol. The LANai processor cannot access the host memory directly; instead it must use the host-to-LANai DMA engine to read and write the host memory. The internal bus clock runs at twice the CPU clock letting the two DMA engines operate concurrently.

3 Reliable Communication

3.1 Background

Most high-level communication APIs provide reliable communication to the user by implementing retransmission in one of the software layers of the protocol stack. Stream sockets, for example, are typically built on top of TCP/IP with reliable end-to-end connections implemented in the TCP layer. This design is suitable for wide area networks since network connections go through intermediate host systems and packets may need to be stored for a long time before they are acknowledged. However, this results in high overhead for TCP; highly tuned implementations add over a hundred microseconds of overhead.

A system area network (SAN) is different from a local or wide area network. A SAN typically has a small diameter (a few feet) and is used to connect computer systems to form a high-performance server. Applications on such networks of computers require low-latency and high-bandwidth communication. A key issue that arises is where and how to provide reliable communication in the system so that high-level APIs can be implemented without compromising performance.

In communication systems, the fault model refers to the level of reliability provided. Previous work has shown that the fault model of a communication system is an important issue that requires attention. A strong fault model simplifies substantially application design and results in a usable system. Systems with weak fault models can be very difficult to program, even for relatively simple applications. Ignoring error handling in the network layers and leaving it to the applications requires substantially more effort on the programmer's side. In general it has been argued that the communication subsystem should provide reliable communication for most cases.

There are two approaches to reliable communication: (i) High performance systems provide reliable communication at the lowest possible level, in hardware. In these systems the network interfaces are responsible for providing the user with reliable communication channels. Communication systems that are built on top of these interfaces treat network errors as catastrophic. This approach works well as long as network errors are extremely rare or when the packet retransmission is handled by the hardware (like S-Connect [11]). However, providing reliability in the physical layer leads to expensive hardware and limits the design choices significantly. (ii) In local and wide area networks, reliable communication is implemented in software in some layer of the communication stack (e.g. TCP) Each approach is targeted for a set of specific needs. In local and wide area networks, higher performance penalties, non-uniform commodity hardware and cost limitations allow and demand solutions at higher software layers. In high performance networks, the same factors lead to very efficient, custom and highly tuned solutions in hardware.

With the emergence of high performance commodity workstations and networks it is possible to provide supercomputer performance on clusters of high-end systems, composed completely of commodity parts. A key issue towards that direction has been to provide the users with very efficient, user level communication libraries. In these libraries, support for reliable communication does not fall in any of the two previous categories. Although error rates in the network are very small, errors do happen and as a result the hardware does not provide reliable transmission. For Myrinet however, errors are observed relatively often (on a daily basis) and cannot be ignored. These new, user–level communication libraries strive for high performance and integration with a traditional fault tolerance mechanism can compromise performance. Thus, the decision of where to incorporate fault tolerance is critical for system complexity and performance.

Reliable communication can be provided in a user-level communication system at different levels. It can be incorporated in either the user–level library that runs on the host, or in the part of the library that runs in the network interface as firmware. We ruled out the design choice of implementing reliable communication at the user-level library. The main reason is that a retransmission protocol requires the sender process not to touch its send buffer until the acknowledgment is received. The sender has either to wait for the acknowledgment, copy the send buffer to a system buffer, or use copy-on-write for the send buffer. The overhead of these design choices is significantly high.

In this work we argue for providing support for reliable communication in the network interface code of the communication library. Implementing retransmission at this level takes advantage of the existing buffering scheme for outgoing packets on the network interface and eliminates the need for the sender to wait for acknowledgments or to copy send buffers. Another reason to provide reliable communication at the data link level is that system area networks are simpler than local or wide area networks. Wide area networks operate in a store-and-forward fashion, whereas in a system area network, a packet is typically routed through switches from a single sender to a single receiver without being stored and forwarded by intermediate nodes.

A lot of work has been done in providing reliable communication in traditional, kernel-based, communication systems and especially TCP/IP. However, this is the only work we are aware of, that argues for and incorporates reliable communication in the network interface, in a user-level communication system for SAN networks. AM [10] support reliable communication in the user-level library by performing one copy. The Nectar system [3] used a general purpose processor in the network interface to run the TCP/IP protocol stack.

3.2 Reliable Communication in the Network Interface

Our scheme deals with transient network failures and provides applications with reliable communication at the data link layer. The goal is to tolerate CRC errors, corrupted packets, and all errors related to the network fabric; links and switches can be replaced on the fly.

3.2.1 Fault Model

We distinguish failures in two orthogonal ways. First, a failure can be either a *network failure* or an *end failure.* Network failures are related to the network interconnect. This includes all links and switches that are used in the network. End failures are the ones that are related to process and node failures. Second, an error can be either transient or permanent. A failure is transient, if the failing component becomes operational after a short amount of time. Otherwise, it is permanent. All failures are considered transient first and, if they persist, they are categorized as permanent. *VMMC-2* deals only with transient network failures.

The fault model adopted in *VMMC-2* is constrained both in terms of design and implementation by many factors. Fault tolerance should not be achieved at the cost of reduced performance. There is limited buffer space on the Myrinet network interface (currently 1 MByte) and the processor on the network interface is a relatively slow processor. A complicated protocol would increase the occupancy of the processor and impact performance. The network we assume is a LAN within a building or other facilities of comparable size. The average meantime between failures is rather big (in the order of days).

As mentioned *VMMC-2* deals only with transient network errors. If transient errors become permanent, the remote node is declared unreachable, the state that is related to the remote node is cleared and the user is notified at the next send (or other) operation. Packets that cannot be delivered to the remote node are dropped. In this case the user needs to reestablish the mappings in order to resume communicating with the remote node. This means that the user does not know exactly which packets were delivered and which not. The semantics are the same as in TCP/IP.

3.2.2 Implementation

VMMC-2 implements a simple retransmission protocol at the data link layer–between network interfaces. Our scheme buffers packets on the sender side and retransmits when necessary. Each node maintains a retransmission queue for every node in the system. The available buffer space is managed dynamically so that no buffers need to be reserved for each node. Each packet carries a unique (for the sender-receiver pair) sequence number. Sequence numbers and retransmission information is maintained on a per-node and not per-connection basis. The receiver acknowledges packets. Each acknowledgment received by the sender, acknowledges (and frees) all previous packets up to that sequence number. There is no buffering at the receiver. If a packet is lost, all subsequent packets will be dropped. However, if a previously acknowledged packet is received again, it is acknowledged. In the current implementation, there are no negative acknowledgments for lost packets.

The retransmission scheme will always deliver packets in the presence of transient network failures. If there are no buffers on the sending side, the senders will back off and wait until space is available on the network interface. The receive side can always receive or drop packets, since the sender will retransmit if packets are not acknowledged.

The number of buffers available at the send side affects system performance. As mentioned above our network interface cards have 1 MByte of SRAM, which leaves us with more than 100 4 KByte-buffers for buffering in the retransmission scheme. These buffers are used dynamically so that no buffers need be reserved for each node.

Although the retransmission scheme will never fail, we should note that malicious processes can temporarily (for a few seconds) hurt the performance of the system by needlessly occupying buffers until the system recovers the buffers with a timeout mechanism. This is not a matter of correctness, but rather fairness in the system.

Acknowledgments can be sent either explicitly or piggy-backed to messages that are transferred in the opposite direction. Piggy-backing reduces traffic and overhead at the LANai significantly, but is useful only when messages are sent in both directions. One way traffic always needs explicit acknowledgments. Explicit acknowledgment messages do not carry sequence numbers and they are not acknowledged. If a packet is lost and the sender does not receive an acknowledgment for it, it will timeout and retransmit the lost and all subsequent packets.

We use two methods for deciding when explicit acknowledgments need to be sent. The first one, specifies that an explicit acknowledgment will be sent every n unacknowledged packets. The selection of n is very important to system performance and scalability. In the case of one way traffic, a large n would require a lot of buffering on the sending side. On the other hand, a small value of n generates a lot of extra traffic; more importantly it increases the occupancy of the network interface which must process the frequent acknowledgements.

The second method requires that the sender specify when the receiver needs to send an acknowledgment. If there have been no piggy-backed acknowledgments, and the available buffer space in the sender is reduced below a threshold, then the sender notifies the receiver with a bit in the packet header, that the receiver needs to acknowledge the packets it received, so that buffer space can be recovered in the sender. We implemented and experimented with both methods and we present results in Section 5.

Ordering of packets is maintained with 16 bit sequence numbers. The system deals with wrap around in sequence numbers. We define the boolean function MoreRecent $MR(x, y)$

$$MR(x, y) = ((x > y) * (x < y + c)) + ((x < y) * (x < y - c))$$

that specifies when a packet with sequence number x is strictly more recent than a packet with sequence number y. c is a constant that depends on the amount of buffer space available in the system and specifies the biggest possible distance between the sequence numbers of any two packets (between a pair of nodes) that are alive in the system at any time. A packet is considered *alive* if it has been sent and the corresponding buffer is not yet reclaimed at the sender.

Since the LANai processor is relatively slow, it is desirable to remove costly operations from the critical path. A number of implementation optimizations are used to distribute the costs uniformly in the execution path in the LANai.

3.3 Performance

Table 1 shows the basic communication performance of *VMMC-2* with and without support for fault tolerance. As mentioned previously, the frequency at which acknowledgments are sent is an important parameter in the system. Table 1 shows also the performance of the system for different acknowledgment frequencies. In the current implementation, we acknowledge every sixth packet (explicitly or with a piggy-backed message) and we use timeouts to handle boundary cases. More frequent acknowledgments reduce the performance of the system because of the extra traffic they generate and the extra processing needed at the sender of each packet. Less frequent acknowledgments generally help, but increase buffer requirements in the system. Piggy-backing reduces the effects of the extra communication, but is useful only in bidirectional communication patterns. Experiments show that it increases bidirectional bandwidth by more than 15%. The second approach we use for determining when explicit acknowledgments are sent, performs the same as our best selection for the fixed threshold method. We see that support for fault tolerance can be provided at low cost. Latency increases by about 2μs (from 11.1μs to 13.4μs) and bandwidth decreases by 2%-10%. The additional overhead comes from moving messages between queues and handling acknowledgments.

# packets/ack	NO RC	1	2	4	6	8	10	units
Latency	11.1	23.93	18.29	14.86	13.4	13.4	13.4	μs
Unidirectional b/w	97.61	82.08	89.97	92.11	92.63	92.81	93.01	MBytes/s
Bidirectional ping-pong b/w	99.39	80.62	87.37	89.54	90.05	90.26	90.39	MBytes/s
Bidirectional simultaneous b/w	91.87	80.30	88.42	89.54	90.77	91.05	91.10	MBytes/s

Table 1: *VMMC-2*, Basic performance numbers with varying ack window. The default value is one acknowledgment every six packets

4 Support for Dynamic System Configuration

4.1 Background

Another important part of a communication system is the way routing is performed. Although a lot of work has been done in routing for multiprocessor networks, routing for emerging SAN networks is mostly unexplored. In traditional multicomputer networks with fixed topologies the routing algorithm is usually embedded in the network hardware. Each packet carries a destination address, which is enough to determine the routing path, given the fixed topology of the network.

In networks however, where the topology is not fixed and changes dynamically, other forms of routing are necessary. In these cases, the communication system supports some form of addressing remote nodes. In general, there are two main methods for delivering packets to their destination. First to have each node in the network know which link each incoming packets should be sent to. These networks require that nodes are intelligent entities that can take decisions based on routing tables and information in the packet headers. Second, to have each packet specify the full path to the destination node upfront. Whenever the packet reaches an intermediate node in the network, its header specifies which link this packet should be directed to. This second form of routing is known as *source routing* and uses simple switches to perform routing inside the network.

Myrinet supports source routing, with the routes being specified in each packet header by the sender. The routing components in the network are simple and fast switches. However, communication systems that are built on top of this type of networks need to be able to determine the routes to any node in the system. This problem has two parts. First, to figure out a graph that represents the topology of the network and second, given a topology, to compute routes. We refer to the first problem as the *mapping problem.* There has been a lot of work in computing routes with specific properties from network topologies [12, 5, 8]. One of the most important properties is that routes should be deadlock free. In this section we present a new method for solving the mapping problem, and we discuss how to solve the problem of deadlock-free routes. In the following paragraphs we discuss some related work in solving the mapping problem for user–level communication systems in SAN networks like Myrinet and then present our solution.

The first solution to the mapping problem is given by Myricom [2]. They use an algorithm that is implemented on the network interface and is integrated with the communication library. This algorithm selects a node as the master node in the system. The master node uses polling to determine the identity of each node in the system and to figure out the topology of the network. The algorithm has to be able to detect cycles and multiple paths to the same node. After the graph that represents the network topology has been determined, it is sent to all nodes in the system. Each node then computes *deadlock-free* routes to all other nodes using the UP*/DOWN* algorithm [12][1]. The Myricom mapping algorithm runs periodically and maps the full network, in the presence of arbitrary traffic in the network.

A second approach is described in [9], where the topology of the network is determined with a different algorithm. Routes are again computed using the UP*/DOWN* algorithm. Their algorithm also maps the full network, but runs on the host processors (as a normal Active Messages program) and requires that there is no other traffic in the network during the mapping phase.

Most other user–level communication systems that are implemented on top of Myrinet use either static maps or the algorithm provided by Myricom to solve the mapping problem. When static maps are used, the user specifies the topology of the network beforehand. After the communication system is started, no changes in the topology are permitted. Reconfiguring the system (changing the topology, or adding and removing nodes) requires a system shutdown and restart. As the number of nodes in a system increase, this solution is not adequate since it is likely that not all the nodes will be operational all the time. Moreover, maintenance of the system requires all the nodes to be brought down, the system to be partitioned and then part of the system to be brought up again. Clearly, some form of dynamic determination of the network topology is needed.

4.2 Support for Dynamic System Configuration in the Network Interface

We take a different approach in solving the mapping problem. Note that in the two previous systems, the map of the full system is constructed each time the algorithm runs. We use an algorithm where routes to each node in the system are determined dynamically, on demand. The basic block in our algorithm is the Myricom approach for polling nodes of the network for their identity. At startup in our system, each node has no information about the topology of the network. Whenever a packet needs to be sent to a node, the network interface firmware starts to poll the nodes in the network, trying to find the destination of the packet. After the destination is found, a route is constructed and stored in a cache, both in the sender and the receiver. The topology of the network is discovered and the routes are built dynamically, on demand.

This approach has many advantages. Whenever there is a change in the topology of the network, there is no need for the full network to be remapped. If a node is added, it will be discovered the first time a packet is sent to it. If on the other hand a node is removed, when a packet is sent, after the retransmission mechanism fails to deliver the packet the node will be declared unreachable and all its state (including the route) will be removed from the cache in the sender. This allows for nodes to be moved in the network transparently. Also the topology can change in arbitrary ways during system operation. Routes to nodes that change relative position will be invalidated and new ones will be computed. Of course the identity of each node needs to remain the same in the network, regardless of each location.

The algorithm deals correctly with packets from previous node or buffer incarnations in the system, something not at all trivial. The data link layer synchronizes properly the sequence numbers in abnormal cases, without worrying about packet generations. At a higher level in the system, generation numbers are used to deliver or drop packets depending on whether they match with the currently exported buffers. This allows for packets that are delivered to the wrong node, because of changes in the topology to be detected and dropped. Generation numbers and fixed node identities are used for this.

This mapping scheme needs to be (and is) integrated with the retransmission scheme that is used in the firmware. Sequence numbers need to be synchronized whenever there are changes in the topology. Moreover, since routes are computed on the fly, without the full topology of the network being available, deadlocks can occur. Myrinet however, has a timeout mechanism for deadlock recovery (not avoidance or detection). This mechanism is used to break deadlocks

[1] The UP*/DOWN* algorithm divides the network links in two groups (UP and DOWN) and imposes an ordering between these categories. This ordering is used to compute deadlock free routes.

when they occur. In our case, we take advantage of this mechanism and the retransmission protocol of *VMMC-2*. If a deadlock occurs the receiver will be reset and packets in transit will be dropped. However, these packets will be retransmitted and eventually delivered to the destination.

4.3 Performance

Our approach for discovering the network topology and computing routes, does not affect the common path in the communication system and thus does not affect performance when the system topology does not change.

5 Overall *VMMC-2* Performance

In order to understand the performance of *VMMC-2*, we measured the communication latency and bandwidth with micro-benchmarks, and analyzed *VMMC-2* with the logP model [4]. Figures 1 and 2 show the performance of *VMMC-2*.

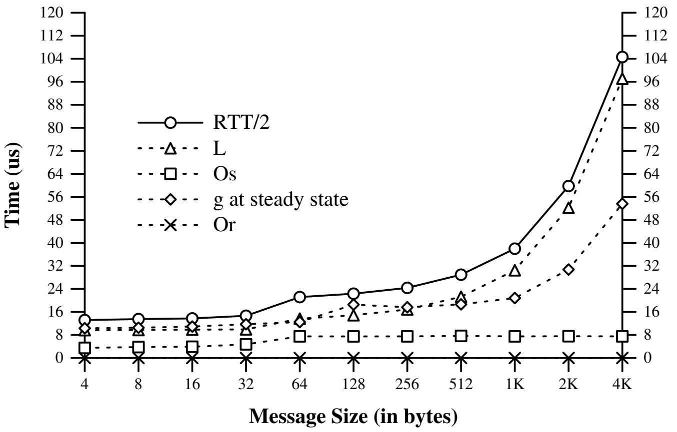

Figure 1: *VMMC-2*, logP numbers

We characterize the performance of our system using the five parameters of the logP model: $(\frac{RTT}{2}, L, O_r, O_s, g)$. $\frac{RTT}{2}$ is the one-way host to host latency for a ping-pong test. O_s, O_r are the host overheads of sending and receiving a message respectively. L is the one-way network latency between network interfaces. Finally g (gap) is the time between two successive sends and describes how fast messages can be pipelined through the system.

Besides the logP characterization we also run three bandwidth tests: *Unidirectional* is the bandwidth of a one way transfer. Data is flowing in one direction only and the sender does not need to wait for the receiver to acknowledge a message before it sends the next one. In the *Bidirectional ping-pong* benchmark data flows in both directions in a ping-pong fashion. After sending a message, each sender waits until it receives a message before sending again. One of the nodes starts as a sender and the other as a receiver. *Bidirectional simultaneous* is similar to the bidirectional ping-pong benchmark, only that both nodes start as senders. This benchmark can potentially use all three DMA engines of the network interface, depending on how the Myrinet firmware is written.

Figure 2 shows results for these three types of bandwidth. Bidirectional ping-pong bandwidth grows slower than the other two types because there is only one message in the network at any given time, so pipelining is less effective for short messages. For longer messages, data is transfered in 4 KBytes packets and unidirectional bandwidth catches up with the other two tests. The maximum bandwidth is in the range of 90-93 MBytes/s for all the tests. The 4 KBytes message bandwidth is about 44 MBytes/s for the bidirectional ping-pong test, and about 80 MBytes/s for the other two tests.

Performance of Sockets: The implementation of Sockets on top of *VMMC-2* takes advantage of the reliable communication support in the communication interface and the absence of data copies and achieves a latency of 20μs for short messages and a maximum bandwidth of 84 MBytes/s.

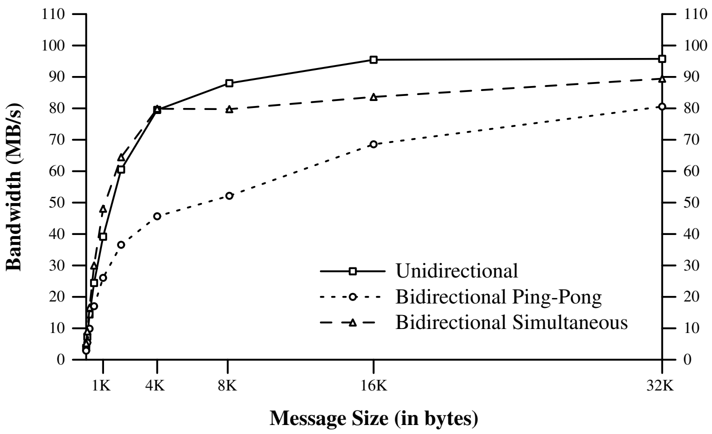

Figure 2: *VMMC-2*, Bandwidth numbers

6 Conclusions

Commodity Systems Area Networks provide performance comparable to traditional multicomputer networks. However, they do not provide the user with reliable communication at the physical layer. This leaves the burden of dealing with errors to the communication library or the application writer. In this work we argue for providing reliable communication in user–level communication systems, in the firmware of the network interface, at the data link layer. This approach takes advantage of the buffering in the network interface to eliminate extra copies. It has a very small impact on performance less than 2μs for latency and less than 10% for all different types of bandwidth. We also present a new solution to the *mapping problem* that takes advantage of the retransmission mechanism in the network interface to simplify certain tasks. This solution, determines the topology of the network dynamically on demand. It allows for changes in the configuration of the system during normal operation. This is very useful for emerging system area networks that are not as tightly coupled as previous multicomputer networks.

We implemented and evaluated the cost of this approach in *VMMC-2*. The overall communication performance of the *VMMC-2* implementation, including the retransmission mechanism and the dynamic mapping features is comparable or better than the performance of systems that do not provide this functionality. Its one-way latency is 13.4 μs for a small messages and it achieves an one-way bandwidth of 80 MBytes/s for 4 KByte messages and over 90 MBytes/s for messages larger or equal to 16 KBytes. Our implementation of stream sockets on top of *VMMC-2* has a one-way latency of 20 μs and peak bandwidth of over 84 MBytes/s.

References

[1] A. Basu, V. Buch, W. Vogels, and T. von Eicken. U-net: A user-level network interface for parallel and distributed computing. *Proceedings of the 15th ACM Symposium on Operating Systems Principles (SOSP), Copper Mountain, Colorado*, December 1995.

[2] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 15(1):29–36, Feb. 1995.

[3] E. Cooper, P. Steenkiste, R. Sansom, and B. Zill. Protocol implmentation on the nectar communication processor. In *Proceedings of the ACM SIGCOMM'90 Symposium*, Sept. 1990.

[4] D. Culler, L. Liu, R. P. Martin, and C. Yoshikawa. LogP performance assessment of fast network interfaces. *IEEE Micro*, 1996.

[5] W. Dally and C. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

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

[7] C. Dubnicki, A. Bilas, K. Li, and J. Philbin. Design and implementation of Virtual Memory-Mapped Communication on Myrinet. In *Proceedings of the 1997 International Parallel Processing Symposium*, pages 388–396, April 1997.

[8] C. J. Glass and L. M. Ni. The turn model for adaptive routing. *Journal of the ACM*, 41(5):874–902, Sept. 1994.

[9] A. M. Mainwaring, B. N. Chun, S. Schleimer, and D. S. Wilkerson. System area network mapping. In *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 116–126, Newport, Rhode Island, June 22–25, 1997. SIGACT/SIGARCH and EATCS.

[10] A. M. Mainwaring and D. E. Culler. Active Message applications interface and communication subsystem organization. Technical Report CSD-96-918, Computer Science Division, University of California at Berkeley, 1995.

[11] A. G. Nowatzyk, M. C. Browne, E. J. Kelly, and M. Parkin. S-Connect: from networks of workstations to supercomputer peformance. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 71–82, New York, June 22–24 1995. ACM Press.

[12] S. S. Owicki and A. R. Karlin. Factors in the performance of the AN1 computer network. Technical Report 88, DEC System Research Center, 130 Lytton Ave., Palo Alto, CA 94301, June 1992.