

# Tolerating Network Failures in System Area Networks

Jeffrey Tang  
Department of Computer Science  
University of Toronto  
Toronto, Ontario M5S 3G4, Canada  
jtang@cs.toronto.edu

Angelos Bilas  
Dept. of Electrical and Computer Engineering  
University of Toronto  
Toronto, Ontario M5S 3G4, Canada  
bilas@eecg.toronto.edu

## ABSTRACT

In this paper, we investigate how system area networks can deal with transient and permanent network failures. We design and implement a firmware-level retransmission scheme to tolerate transient failures and an on-demand network mapping scheme to deal with permanent failures. Both schemes are transparent to applications and are conceptually simple and suitable for low-level implementations, e.g. in firmware. We then examine how the retransmission scheme affects system performance and how various protocol parameters impact system behavior. We analyze and evaluate system performance by using a real implementation on a state-of-the-art cluster and both micro-benchmarks and real applications from the SPLASH-2 suite.

## 1. INTRODUCTION

Recently there has been a lot of progress in providing low-latency, high-bandwidth access in commodity clusters by using system area networks (SANs). These advances in the interconnection network technology both at the hardware and architectural levels have resulted in efficient SANs [6, 16] and higher, user-level communication protocols [27, 11, 3, 17, 32, 31]. System area networks usually connect high-end workstations within a computer room or building and have evolved from the interconnection network technologies used in parallel systems. They are widely used in commodity clusters, which are currently being deployed in a wide spectrum of areas, e.g. as replacements to more expensive parallel computing [5] and storage systems [33]. Despite the improvements on the performance side, very little work has been done on tolerating transient and permanent failures in the SAN interconnects. The underlying assumption is that transient errors, such as packet corruption, as well as permanent failures, such as switch errors, are rare events. As these networks are finding commercial applications, besides performance, fault tolerance and availability are also becoming important issues. However, due to cost and complexity reasons, system area networks do not necessarily provide reliability at the hardware level. Moreover, most of the recent work in system area networks has not studied the performance implications of tolerating network errors. Thus, little is known today about these aspects of SANs, and especially their commercial versions [18, 6, 16] and the effects on real applications.

In this paper we investigate how failures can be toler-

ated in system area networks. We demonstrate how modern network interface controllers (NICs), such as Infiniband adapters, can incorporate in firmware a retransmission protocol to tolerate transient network failures and an on-demand mapping scheme to discover alternate routes when permanent network failures occur. Our retransmission scheme avoids all copies, has low memory and computational requirements, and imposes low overhead in the common path. Our on-demand mapping scheme only uncovers paths to nodes where messages need to be sent, and imposes overhead only when re-mapping is necessary. Both of these schemes are transparent to any layers above the network interface in general and to the application layer in particular.

Besides proposing the above solutions, this work also attempts to answer the following performance-related questions: What is the performance impact associated with providing reliability in firmware? What is the effect of error rate on system performance? What are the most important parameters in the reliability protocol, what is their effect on application performance, and what values are the best compromise between fast recovery and low overheads in the absence of errors?

We implement our schemes on a state-of-the-art cluster of Intel-based PCs with a programmable system area network. We run both micro-benchmarks and real life applications from the SPLASH-2 shared-memory application suite. We find that: (i) Adding reliable transmission at the firmware level increases one-way latency for small messages by at most  $2.1\mu\text{s}$  for message sizes up to 64 bytes (at most 20% increase) and reduces bandwidth by less than 4% for all message sizes above 4 KBytes. (ii) Our retransmission scheme is very robust. Application performance degrades significantly only at very high error rates of  $10^{-3}$  and upwards. Application performance is practically unaffected with all but these very high error rates. (iii) For the specific retransmission protocol parameters we find that: The best value to use for the retransmission timer in setups similar to our system is 1ms. Longer timer intervals result in slower recovery times and shorter timer intervals result in higher communication overheads. The NIC send queue size has little effect on applications performance. Queue sizes of 4 or greater work well in all applications and error rates that we examine. (iv) Our on-demand dynamic mapping scheme is orthogonal to the network mapping algorithm and does not introduce any overhead in the common case.

The rest of the paper is organized as follows. Section 2 provides the necessary background and discusses related work. Section 3 describes our experimental platform. Section 4 presents the protocol design. Section 5 discusses our evaluation methodology. Section 6 presents our results. Finally, Section 7 draws our conclusions.

## 2. BACKGROUND AND RELATED WORK

In this work we examine how transient and permanent *network* failures can be tolerated in System Area Networks (SANs). Previous work has examined these problems in the context of other types of networks, mainly local area networks (LANs) [30, 28] and parallel interconnection networks [9, 21, 1]. However, existing solutions for other types of networks are not directly applicable to SANs. Modern SANs are different from both of these types of networks. Compared to LANs: SANs offer much higher bandwidth and lower latency. Applications running on SANs are much more demanding in terms of network resources and are less tolerant to latencies. SANs typically have smaller diameters (a few feet) and use source routing rather than store-and-forward for message exchanges. The error rate in SANs is lower, resulting in different tradeoffs. SANs exhibit lower round-trip latencies and thus require less buffering. Compared to parallel interconnection networks, SANs exhibit the reciprocal behaviors. In addition, SANs support arbitrary topologies and have lower cost.

Research projects in system area networks and user-level communication systems have taken different approaches in dealing with network errors. The authors in [20] demonstrate the need for providing reliability in lower layers (the NIC) as opposed to the library or the application level. A preliminary version of our schemes was presented in [11] in the context of VMMC-2. However, that work focuses on support for streaming communication as opposed to tolerating network failures. AM-II [8] provides reliability guarantees similar to our system using acknowledgments and timeouts. However, their approach differs in a number of ways: The retransmission scheme is implemented at the host (library) level. Timers are managed on a per packet basis. Sending a packet schedules a timer event, and receiving an acknowledgment stops the associated timer. In contrast, we maintain only one timer in our system for all packets. AM-II provides receiver-side buffering combined with negative acknowledgments (NACKs) to provide flow control. Our scheme does not use NACKs. PM [31] provides flow control using a modified ACK/NACK mechanism. In this scheme, the sender does not free any packets until a positive acknowledgment is received, and the receiver is free to discard incoming data, i.e. when its buffer pool is exhausted. The sender retransmits upon receiving negative acknowledgments. This protocol guarantees in-order message delivery. However, PM assumes the network to be reliable and the acknowledgments to always reach their destination. BIP [15] and FM [27] assume that the underlying network is reliable and treat network errors as catastrophic. The authors in [4] present LCI, a Low-level Communication Interface, that supports reliability and multicast. They study the relationship of communication protocol decomposition (between the NIC and the host) with application character-

istics. Similar to our work, they find that NIC support for both reliability and multicast performs best for the applications they examine. However, although the overhead of reliability on small-message latency is similar to our solution, the impact on bandwidth performance is significantly higher. Maximum bandwidth performance in LCI is about 70MBytes/s compared to 120 MBytes/s in our scheme. Finally, in our work we also examine the retransmission protocol characteristics with respect to various system parameters and the network error rate. BDM/Pro [17] provides reliability in the communication library. All sent packets are retransmitted if the acknowledgments do not arrive within a certain amount of time. The receiver only acknowledges groups of  $N$  packets as well the last packet of each message. However, BDM/PRO buffers data on the host side rather than on the NIC as in our approach. When packets must be retransmitted, they are re-DMAed from the host memory. The Virtual Interface (VI) specification [12] defines three levels of reliability: Unreliable delivery, reliable delivery, and reliable reception. VI NICs are only required to support the unreliable delivery mode. Our work shows that reliability in VI networks can be implemented at the NIC level with minimal overhead. Similarly, Infiniband [18] specifies reliability levels for different types of communication and provides a number of guidelines on how these should be implemented. In our work we design, implement, and evaluate a firmware-level retransmission scheme that serves as a guide for similar implementations in Infiniband NICs.

Tightly coupled multiprocessors have traditionally implemented reliability at the hardware level. The hardware scheme used in most Cray systems [9] is the single bit error correction, double bit error detection (SECDED) scheme. SECDED is more appropriate for implementation in the memory controller. SANs usually operate at the PCI bus level and packet based schemes are more appropriate as opposed to memory-level based schemes. The interconnect employed in the Origin 2000 System [21] is based on the SGI SPIDER router chip [14]. To achieve high availability, the interconnect fabric isolates each module so that a failure in one module does not affect other modules. Furthermore, all high-speed router and I/O links are protected by a full cyclic redundancy check (CRC) code and a hardware link-level protocol that detects and automatically retries faulty packets. The flexible routing network supports multiple paths between nodes, partial population of the interconnect, and hot swapping of links for servicing faulty hardware. Their hardware scheme is similar to our approach in that corrupted or lost packets are retransmitted based on a Go-Back-N sliding window protocol. In contrast, however, our implementation assumes a slower processor on the NIC and a more loosely-coupled interconnect, e.g. higher system latencies and arbitrary topologies. The micro-channel adapter interconnects nodes to switch boards in IBM SP2 [1] and generates a CRC code for each packet. To tolerate permanent errors, each switching element is shadowed by a duplicate switching element. Each switch also contains at least one stage more than necessary for full connectivity, and this extra stage guarantees that there are at least four different paths between every pair of nodes. The redundant paths also reduce congestion

in the network. Furthermore, the error-detection support in hardware is coupled with error recovery capability in the communication software, which supports end-to-end packet acknowledgment and retransmission if an acknowledgment is not received promptly.

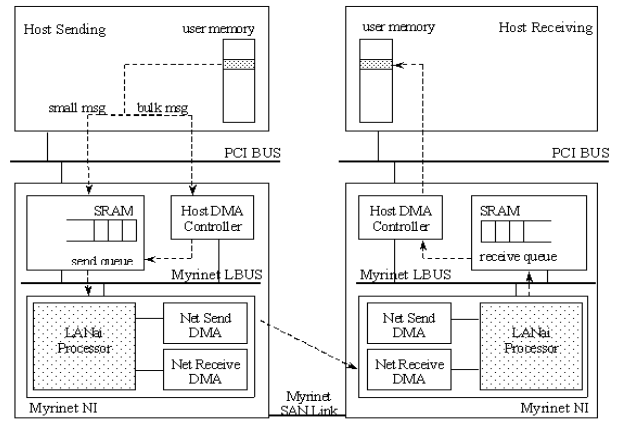
With respect to tolerating permanent network failures, the most common approach has been to either use routing schemes that statically account for redundant routes and use an alternative route in case of permanent failures [2, 1] or to fully re-map the network when a failure occurs [6, 28, 22]. Previous research in SANs aimed at finding cost-effective, efficient algorithms for determining full network maps and deadlock-free routing paths in SANs. In some cases it also considered the quality of the generated routes. The authors in [6, 28, 22] provide schemes for discovering full network maps and use the UP\*/DOWN\* algorithm to determine deadlock-free routes. UP\*/DOWN\*, proposed in [10, 29], is a popular algorithm for finding deadlock-free network routes. However, the selected paths are generally not shortest paths. The authors in [13] examine several variations of the UP\*/DOWN\* algorithms by exploiting the fact that more than a single route is possible for a given source-destination pair. Virtual Channels [23] (VCs) can be used to deadlock-free routing algorithms and to increase link utilization. The authors in [24] study the sensitivity to failures of two routing algorithms, UP\*/DOWN\* and minimal adaptive routing. These two algorithms are also examined in the context of fiber channel storage area networks [25]. Our work leverages the derived mapping algorithms. It takes advantage of the retransmission scheme we design as a deadlock recovery mechanism to avoid computing full network maps and deadlock-free routes that greatly complicate the network mapping process. Although our scheme has the potential of improving on the quality of routes, in this work we do not investigate this any further.

### 3. EXPERIMENTAL PLATFORM

Our experimental environment consists of Intel-based PCs connected with a Myrinet network [6]. Each PC is equipped with two Pentium II processors clocked at 450MHz and 512 MBytes of main memory. The communication layer we use in the system is VMMC [11].

#### 3.1 Myrinet

Figure 1 shows the block diagram of the Myrinet NIC architecture. In this work we use the Myrinet M2M-PCI64A-2 adapters. The PCI-based NIC is composed of a 32-bit control processor (LANai 7) with 2 MBytes of SRAM. This memory is used for network buffers as well as for firmware code and data. The LANai 7 is clocked at 66 MHz and executes a loadable Myrinet control program (MCP). The MCP controls the NIC resources, e.g., DMA engines, and implements the low-level communication protocols. Each NIC has three DMA engines: two for transferring data between the network and the SRAM and one for transferring data between the SRAM and the host main memory over the PCI bus. The DMA engines can be controlled either by the host or the LANai processor. The host can also access the SRAM using programmed I/O. Each network link is full duplex with 1.28 GBits/s bandwidth in each direction.



**Figure 1:** Message exchanges in VMMC. The arrow points to the direction of data flow, for a typical send operation.

All the switches we use are full crossbars. Myrinet uses wormhole routing and the entire route needs to be stored in each packet header.

#### 3.2 Virtual Memory Mapped Communication

The communication model used by VMMC provides protected user-level communication between the sender's and the receiver's virtual address spaces. Before communication can take place, the receiving process exports areas of its address space where it is willing to accept incoming data with a set of permissions. The sending process must import remote buffers before directly depositing data to remote memory. Communication is protected in that the exporter can restrict the processes and hosts that can import a buffer.

Figure 1 illustrates how messages are exchanged in VMMC. On the send side, messages are transferred to the NIC in one of two ways: programmed I/O for small messages ( $\leq 32$  bytes) and DMA for medium and large messages ( $> 32$  bytes). Programmed I/O involves the host CPU transferring data directly to the NIC address space as opposed to data movement by the DMA engine. Messages larger than 4 KBytes are segmented into chunks by the MCP. On the receive side, the packet is first deposited in a receiving queue on the NIC. Since the NIC also maintains the translations between physical and virtual addresses of receive buffers [11], the MCP DMA's the incoming packet into host memory directly without having to interrupt the host processor.

#### 3.3 Error Handling

The transient network errors that can occur are packet corruption and packet loss. Different types of interconnects may experience different sources for these errors. In Myrinet the sources of these errors are: (i) Hardware problems that can cause both packet corruption and loss. Although the hardware error rate in Myrinet is very low ( $10^{-15}$ ), the possibility of hardware error is still present and cannot be ignored, especially in cases of commercial applications. (ii) Network timeouts that can cause both packet corruption and loss. As a deadlock detection and recovery mechanism, Myrinet provides a user configurable timer (62.5ms to 4s) [6]. If the send path is blocked for more than this

amount of time (for any reason), the sender resets the path. The reset empties all hardware queues along the path and drops any stored packets or parts of packets. (iii) Communication software behavior that should only cause packet loss, e.g., by voluntarily dropping packets. For instance, the communication software may drop packets for flow control reasons.

The Myrinet hardware (and other SANs) provides mechanisms to detect packet corruption errors using CRC checksums. On the send side, the network DMA computes a 32-bit CRC and appends it to the end of each packet. On the receive side, the network DMA computes the CRC of the incoming packet. To detect any packet corruption, the MCP needs to check the two CRCs. Intermediate switches recompute CRCs on the fly to take account for the modified packet headers. Currently, there is no hardware mechanism to detect packet loss.

## 4. PROTOCOL DESIGN

For the purpose of our work we define network failures as any failures that occur on the communication path between a pair of nodes after a packet has left the sending NIC and before it is delivered to the receiving NIC. We consider NIC failures, not as part of the interconnection network, but rather the end-nodes themselves. In our system, end-node failures are dealt at a higher level [7]. Network failures are categorized as either *transient* or *permanent*. Transient network failures are temporary, and the failing components return to normal operation after a fixed and usually small amount of time. Permanent network failures occur when a system component, e.g., link or switch, cannot recover from a failure and usually needs to be replaced. Transient network failures are tolerated with a packet retransmission scheme. Permanent network failures require the existence and discovery of alternate routes in the interconnect. The main challenge is to perform these tasks without introducing high overheads in the common case, when there are no failures, and to minimize the recovery time in case of failures.

To distinguish between transient and permanent failures we simply use a time interval threshold. If a failure on a path between two nodes is remedied (by a successful packet delivery) within this interval the failure is treated as transient. If the failure persists for longer than this threshold, without any successful packet deliveries, the path is considered to have a permanent failure and is marked as invalid.

### 4.1 Tolerating Transient Failures

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. Retransmission can be incorporated either in the user-level library running on the host or in the communication library running on the network interface as firmware. Implementing the reliability protocol on the host may impact application performance significantly as explained later. Our approach to reliable communication is to implement the retransmission method on the NIC. However, providing reliability on the NIC requires careful resource management. The NIC processor is

significantly slower when compared to the host processor. Furthermore, the amount of memory available on the NIC is limited.

In our scheme, each packet is assigned a unique sequence number. After the packet is transmitted to the destination node, instead of freeing the packet buffer, it is placed in a retransmission queue. When the receiver receives a packet, it sends back an acknowledgment. When the sender receives the acknowledgment, it frees the packet buffer. The sender employs a timeout mechanism, where it periodically checks the retransmission queues for packets that have not been acknowledged for an extended period of time. These packets are then retransmitted. Any network error will result in the receiver not acknowledging one or more packets, and the sender retransmitting a set of packets.

#### 4.1.1 Design choices

Within this general scheme there is a wide range of design and implementation decisions that need to be made, based on the semantics of the communication layer (VMMC provides in order, point-to-point delivery), the overhead of implementing each choice, and the behavior of the resulting retransmission scheme in the presence and absence of network errors. In our particular implementation: The full retransmission scheme is implemented in firmware on the NIC. Each acknowledgment carries a single sequence number and acknowledges *all* packets up to (and including) that particular sequence number. Also, to reduce NIC occupancy, there are no negative acknowledgments. On the sender, each remote node has its own retransmission queue. This is a critical issue for system scalability. Using retransmission queues per pair of user processes would result in high resource requirement in the firmware. A similar solution to our work is adapted in Infiniband [18]. When the receiver does not receive a particular sequence number, it will drop immediately *all* subsequent packets from that remote node until it receives the expected sequence number. Thus, there is no packet buffering at the receiver. Upon retransmission, the sender retransmits (in order) *all* packets in the retransmission queue for the particular node. Acknowledgments are not critical, in the sense that they can be dropped. Sequence numbers and retransmission information are maintained on a per-node and not per-connection basis.

These design choices result in a number of advantages: We push functionality towards the NIC with little additional complexity to reduce the performance and complexity impact on higher layers [20]. We take advantage of packet buffering that takes place on the NIC and we do not introduce extra packet copies in the retransmission code. This is not the case in schemes that are implemented on the host side, at the library level. In those cases, either a copy of each packet/message needs to be made or the user is not allowed to reuse the buffer until the acknowledgment is received. Moreover, these overheads always occur in the common path. None of these limitations exist in our scheme. Our implementation is lightweight, requires very little resources, and is appropriate for implementation at the firmware or hardware level. On the send side, storing a packet after transmission, retransmitting a packet, and freeing acknowledged packets involves simply moving

packet buffers between queues (the global NIC transmission queue, the individual node retransmission queue, and the global NIC free queue). The retransmission timer can be relatively long so that a small overhead is imposed. On the receive side, the only additional operations needed are either acknowledging a packet or dropping out of order packets. Acknowledging packets involves sending a simple acknowledgment message including a sequence number. Dropping a packet is a simple dequeue operation.

#### 4.1.2 Optimizations

We apply a number of optimizations on this basic scheme. First, instead of sending explicit acknowledgments, we use piggy-backed acknowledgments in regular data packets in cases of two-way traffic. Explicit acknowledgments are still needed in cases where there is one-way traffic for extended periods of time or the sender will run out of send-buffer space. Second, we do not need to acknowledge every packet. Instead, a single acknowledgment can be used to acknowledge a set of consecutive packets. Upon receiving an acknowledgment, the sender frees all related packets with a single operation. Third, to vary the frequency of acknowledgments we employ sender-based feedback. Each packet, from the sender to the receiver, contains a bit stating whether an acknowledgment is needed so that buffer space can be freed. This approach allows the sender control over how fast its NIC buffers should be released based on the amount of available system resources. When the number of available NIC buffers is small, the sender would request for an explicit acknowledgment so that the buffers can be freed in a timely fashion. When the number of available NIC buffers is adequate, the sender could afford to wait for return traffic and requests for piggy-backed acknowledgments. When the number of available NIC buffers is large, the sender reduces the frequency of acknowledgment requests. These optimizations create a trade-off between the amount of buffer space required at the sender and the frequency of acknowledgments, which we examine later.

## 4.2 Tolerating Permanent Failures

The retransmission scheme can tolerate transient failures but is not able to deal with permanent network failures, such as permanent link and switch faults. To transparently deal with these types of errors we propose and use a simple extension to existing infrastructure. The Myrinet firmware includes a mapping component [6] that can discover the full topology of a system area network by performing a breadth first search. It forms a spanning tree and extracts the routes between all host pairs in the network using the UP\*/DOWN\* algorithm [26].

We propose extending this mapping scheme in two ways: To compute only partial maps of the network and to dynamically compute new routes as they are needed. When a NIC needs to communicate with another NIC or if packets cannot be delivered over an existing route, it starts mapping the network to find a new route to the destination node. The NIC returns to normal operation as soon as a route is found.

If no alternative route to a node exists, the node is labeled as unreachable and any pending packets are dropped. If an alternative path exists, sequence numbers are reset, a

new generation of sequence numbers is started, and communication is resumed. With these extensions it is not clear anymore when a node joins the network, i.e. if this node is a new node or a reincarnation of a previously existing node. The sequence number management scheme clearly distinguishes between packet generations and drops packets in the network that belong to previous generations as soon as a new generation is started.

This approach has a number of benefits compared to mapping techniques that compute full system maps. The effect of permanent errors is localized. When a permanent error occurs, only the affected NICs will invalidate the failing routes and discover new ones (if they exist) on demand. Moreover, while computing each route in the network, there is no need to stop the rest of the traffic in the system. Computing a route happens concurrently with other traffic in the network, localizing the delay mostly to a single pair of nodes. Each node only needs a partial set of routes to the nodes that it is communicating with and does not need to maintain full routing tables. Moreover, whenever a fault occurs, the initial recovery overhead is lower, since only the necessary routes will be recomputed and this cost is amortized over a longer period of time. Furthermore, no overhead is added to the common path (when routes are already computed and no failures occur). Besides tolerating permanent failures, this scheme allows for dynamic network reconfiguration. Nodes can move in the network and routes will be recomputed dynamically as they are needed. There is no need for a central map manager that performs the mapping. Any node can discover routes to other nodes. This is an important feature of the scheme, since requiring a central map manager usually means that additional care needs to be taken to deal with failures of the map manager. Since deadlock-free routes are not needed, the quality of the routes may be improved. However, we do not explore this direction any further and improving the quality of the computed routes by load balancing the induced traffic or in other ways is beyond the scope of this work.

A key issue is that our scheme relies on retransmission for correctness. Since we do not compute the full network map and we do not take into account any existing routes in the system, it is possible that deadlock situations may arise. The final set of routes computed is not deadlock free. Our mapping scheme relies on the retransmission protocol to deal with deadlocks. When there is no progress in a specific path, the sender resets the path to the receiver and clears all hardware buffers. Then, our retransmission protocol will retransmit each packet. Thus, instead of computing deadlock-free routes to avoid deadlocks, we rely on deadlock detection and recovery.

The main disadvantage of our scheme is that since the network is mapped independently by all nodes, work is replicated. This may result in longer times for mapping the full network. Although various techniques, e.g., caching, may be used to improve this, we do not explore this direction any further in this work.

## 5. EVALUATION METHODOLOGY

Our main goal in the rest of the paper is to identify the most important parameters of our retransmission scheme

Parameters	Values			
# NIC Send Buffers	2	8	32	128
Timeout Interval	10 $\mu$ s	1 ms	100 ms	1 s
Error Rates	0	$10^{-3}$	$10^{-4}$	$10^{-5}$

**Table 1:** Range of system parameters studied in this work.

and examine the performance sensitivity as these parameters change. We would further like to use this knowledge and tune such retransmission schemes to incur minimal overheads in the absence of network errors.

## 5.1 Retransmission Protocol

The most important parameters in our retransmission scheme are: The NIC send queue size, the retransmission timeout interval, and the network error rate. Table 1 shows the ranges of the parameter values that we examine.

### 5.1.1 Number of send buffers

The NIC send queue size affects the amount of concurrency and the cost of recovery in the system. Since the NIC send buffers are not released until acknowledgments are received, this parameter determines the number of packets that can be pipelined across the network before the sender blocks due to a lack of send buffers. The amount of memory that is available for send buffers on the NIC can vary significantly, from a few KBytes to a few MBytes or so in the most aggressive, high-end NICs. This memory needs to be shared among the firmware code, data structures, receive, and send buffers. In our system we use NICs that are relatively aggressive with 2 MBytes of memory. Thus, we have the ability to vary the number of send buffers in a wide range of values. The values that we choose are between about 8 KBytes and 512 KBytes (each buffer has a fixed size of about 4 KBytes).

We should mention that receive-side buffering is not as important for this work. We make sure that there are enough NIC receive buffers available so that multiple senders can never overwhelm a receiver. This is based on two features of our system: First, the number of receive buffers available on each NIC is about the same as the number of nodes in the system. Thus, each sender is guaranteed at least one receive buffer regardless of the packet destination. With the current technology, a cluster of SMPs with 128 processors is considered a relative large system [19] and requires at most a few tens of receive buffers. Furthermore, current trends indicate that the NIC memory increases faster than the scaling of the system. Second, the receive path is considerably faster than the send path. In particular, the sender side needs to perform operations such as address translation, preparing packet headers, transferring data from host to the NIC, and managing retransmission buffers. On the other hand, the receiver only needs to move data directly to host memory upon packet arrival.

### 5.1.2 Retransmission timer interval

The retransmission timer imposes a trade-off between the overhead in the common case (no errors) and the recovery time when errors occur. A small timeout interval increases the processing overhead and may lead to false retransmissions in cases of high network contention. A large timeout

interval incurs long recovery delays when a packet is lost. We are interested in determining a timeout interval that recovers well from errors and at the same time introduces little overhead to the common path. Since the minimum round-trip time in our system is about  $16\mu$ s we choose a range of values between  $10\mu$ s and 1s for the retransmission time. The  $10\mu$ s-value, although practically not realistic, is chosen as one extreme of the spectrum to force very frequent retransmissions. The 1s-value is chosen as the other end of the spectrum.

### 5.1.3 Network error rate

Since we do not have access to switches (no firmware), to simulate network errors in a controlled manner, we model network errors by dropping packets on the send side NIC, right before they are injected to the network. At predefined packet counts, the dropping mechanism on the NIC inserts the next packet in the retransmission queue without actually transmitting it onto the network. Since packet is not delivered, the receiver does not acknowledge this and any subsequent packets from this source node. When the retransmission timer expires on the send side, the sender retransmits all unacknowledged packets, starting with the dropped packet. By changing the dropping interval, we are able to control the rate at which packets are dropped. Detection of lost packets on the receive side includes the cost of detecting packet corruption. Thus, dropping packets to simulate network errors (as opposed to, e.g., inserting corrupted packets) incurs the highest possible overhead on the receive side. We perform experiments with varying error rates and contrast our results with the case where no errors occur. To reduce the amount of time it takes to run experiments, we start with error rates of  $10^{-5}$  (i.e. drop one packet every  $10^5$  packets) and increase the error rate by a factor of 10 at each step, i.e.  $10^{-4}$ ,  $10^{-3}$ . Finally, we do not experiment with bursty errors, since high, uniform error rates are a more stressful test.

### 5.1.4 Evaluation workload

We use both micro-benchmarks and real applications. At the micro-benchmark level we use a simple latency test, a ping-pong bandwidth test, and a unidirectional bandwidth test. Unlike the ping-pong bandwidth test, in the unidirectional bandwidth test data flows only in one direction. The sender does not wait for the receiver to acknowledge a message before sending the next one. This type of bandwidth test measures how fast data can be put onto the network.

Application	Problem Size	Other Parameter
FFT	1 M points	18 iterations
RadixLocal	4M keys	5 iterations
WaterNSquared	4096 molecules	15 steps

**Table 2:** SPLASH application problem sizes.

We also use a subset of the SPLASH-2 application suite as modified in [19]: FFT, RadixLocal, and WaterNSquared. FFT performs a six-step FFT algorithm that minimizes interprocess communication. It is a single-writer application in that a given word of data is written only by the pro-

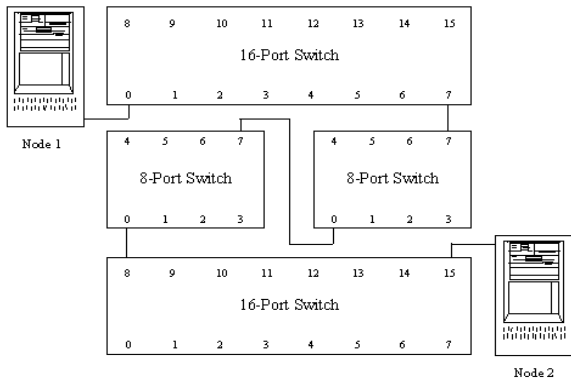


Figure 2: System setup for dynamic mapping.

cessor assigned, and it incurs high communication (bandwidth limited). RadixLocal performs an integer radix sort, and it is a variation of the Radix program from the original SPLASH suite. Radix performs fine-grain accesses to shared data (latency sensitive). The version under study is an improvement over the original version to reduce the irregularity of accesses [19]. WaterNSquared evaluates the forces and potentials in a system of water molecules. It is an  $O(n^2)$  algorithm with a small communication to computation ratio but uses lock synchronization heavily.

We perform the micro-benchmark tests on a pair of nodes connected with a switch and the applications tests on a sub-cluster of 4 nodes with 8 processors total. In all cases we generate enough packets to allow at least ten packets to be dropped at the lower error rate (and more in the higher error rates). In FFT and RadixLocal we modify the programs to perform the computation for more iterations and increase the run time. In WaterNSquared, we increase the problem input size. Table 2 shows the exact input parameters and problem sizes for each application.

## 5.2 Dynamic System Mapping

Our dynamic mapping scheme does not interfere at all with NIC operations when there are no failures. It is only triggered when a node cannot be reached after a number of retransmissions. For this reason we only present basic measurements of the number of messages and the time it takes to map a single node in our system. The mapping time depends on the system topology and network size that is used. We consider our setup to be representative of many real-life SANs used either in clusters that perform parallel computations or in storage area networks. In all these cases nodes are connected in a tree with redundant links to avoid single-points of failures. We configure our system to include four switches: two 16-port and two 8-port, full-crossbar switches. We connect two of the system nodes at various locations of the system, as shown in Figure 2. We present both the time and the number of probe messages it takes to map a single pair of nodes in our setup.

## 6. RESULTS

We explore the entire parameter space as presented in Table 1. However, due to time and space constraints we present results for only a subset of the parameter values.

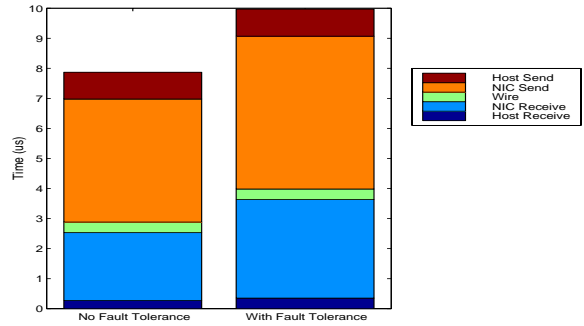


Figure 3: Latency breakdown for 4-byte messages.

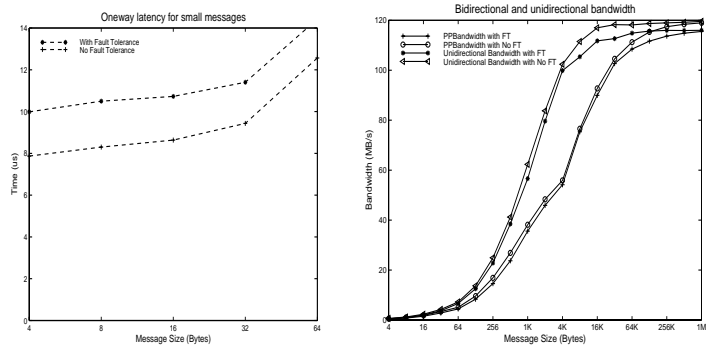


Figure 4: System latency and bandwidth with and without the retransmission protocol.

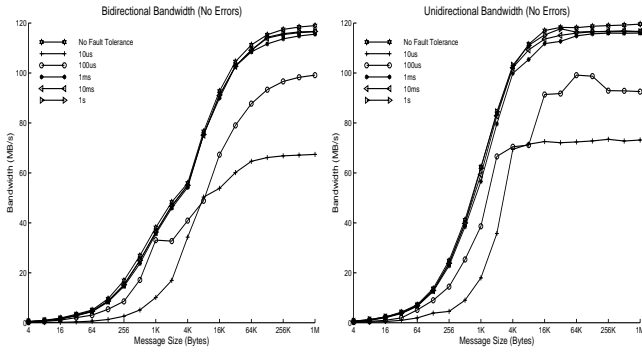
We extract the more important observations and then present the data that support these observations.

## 6.1 Reliable Transmission Protocol

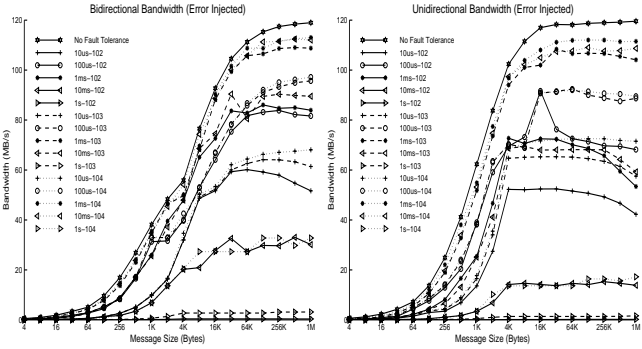
### 6.1.1 Overhead

We find that adding reliable transmission exhibits a 20% overhead in the absence of errors. Figure 3 compares the one-way latency breakdown of a 4-byte message with and without retransmission. The retransmission protocol overhead is divided almost equally between the send and the receive sides, i.e. about  $1.0\mu\text{s}$  in each case. The overhead in the send path results primarily from the management of retransmission queues. The overhead in the receive path results primarily from the processing of acknowledgments. These overheads are fairly low considering that the NIC processor is relatively slow. Overall, the highly-optimized initial latency for a 4-byte message is increased from about  $8\mu\text{s}$  to  $10\mu\text{s}$ . Figure 4 illustrates the cost of the reliability protocol in terms of micro-benchmarks. In particular, the latency overhead is less than  $2.1\mu\text{s}$  for messages up to 64-bytes, and the bandwidth overhead is less than 4%, for all messages sizes above 4 KBytes in both the bidirectional and unidirectional cases. The system bandwidth for large messages is limited by the 32-bit PCI bus at around 120 MB/s.

Finally, we should note that these measurements for the overall overhead of the retransmission protocol have been performed for a retransmission interval of 1ms and a NIC send queue size of 32, since these are the best values for our



**Figure 5:** The effect of retransmission interval on bandwidth with no errors. The NIC send queue size is set to 32.



**Figure 6:** The effect of retransmission interval on bandwidth with errors injected. The NIC send queue size is set to 32. The error rates presented are:  $10^{-2}$ ,  $10^{-3}$ , and  $10^{-4}$ . setup as will be explained next.

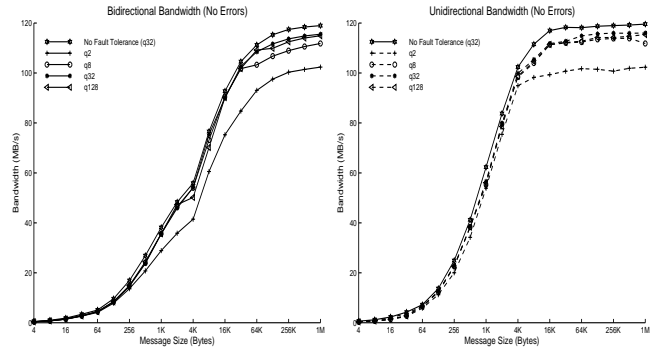
### 6.1.2 Effect of retransmission timer interval

Figure 5 shows the effect of the retransmission timer in the absence of errors and a network queue size of 32. We see that a retransmission interval of  $100\mu\text{s}$  or less incurs a high overhead (bandwidth drops by more than 17% across all message sizes), whereas an interval of 1ms or longer introduces a much lower overhead.

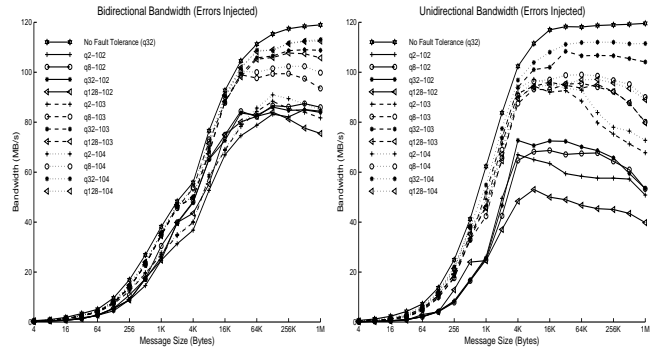
Figure 6 shows the same statistics in the presence of varying errors. The curves in the figure correspond to the various retransmission intervals under selected error rates. We observe that using a timer value of 1ms results in robust behavior as the error rate varies. Even at the high error rate of  $10^{-4}$ , system bandwidth is still within 10% of the case where no network errors occur for message sizes of 4 KBytes and up. As the retransmission interval becomes shorter or longer system bandwidth drops significantly, by more than 18% for  $100\mu\text{s}$  and 72% for 1s, for the same message sizes.

### 6.1.3 Effect of network send queue size

Figure 7 shows the effect of the NIC send queue size in the absence of network errors and a retransmission interval of 1ms. We see that only very small queue sizes have an effect on system performance. Any queue size above 8 results in the close-to-maximum bandwidth. Figure 8 shows that in the presence of errors the same trends are still valid. Any



**Figure 7:** The effect of NIC send queue size on bandwidth with no errors. The retransmission interval is set to 1ms.

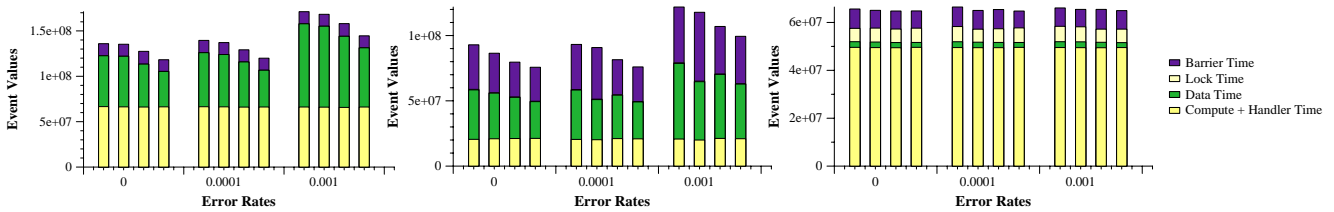


**Figure 8:** The effect of NIC send queue size on bandwidth with errors injected. The retransmission interval is set to 1ms. The error rates presented are:  $10^{-2}$ ,  $10^{-3}$ , and  $10^{-4}$ .

network send queue size above 8 or so results in bandwidth close to the best case when the error rate is  $10^{-4}$  or less. For higher error rates (i.e.  $10^{-2}$ ), however, the performance for large queue size degrades more. In particular, a large send queue size of 128 in the unidirectional bandwidth results in more than 30% reduction in bandwidth. This behavior is the result of sender-based feedback that attempts to reduce the number of acknowledgments when there is plenty of resources (i.e. send buffers) and the fact that the system does not support selective retransmission. As a sender with a large queue size slows down the rate of freeing NIC buffers by not requesting ACKs, the performance degrades faster at higher error rates.

### 6.1.4 Real applications

Figure 9 shows the execution time breakdowns for selected parameters values for each of the three applications we examine. The SVM protocol we use is GeNIMA [5], which leverages NIC support to eliminate all synchronous communication operations and exhibits relatively high latencies and increased network traffic. The results are grouped by error rates, with each group consisting of 4 bars. Each bar represents the application execution time for a particular parameter configuration. Our results are very close to that of the micro-benchmark results. First we notice that WaterNSquared is rather insensitive to most of the parameter changes. This is due to the high computation to communication ratio that it exhibits. Second we notice that



**Figure 9:** FFT (left), RadixLocal (middle) and WaterNSquared (right) execution time breakdowns, grouped by error rates. Each group consists of 4 bars that correspond, from left to right, to the following parameters:  $r100\mu s-q2$ ,  $r100\mu s-q32$ ,  $r1ms-q2$ , and  $r1ms-q32$ .

for error rates up to  $10^{-4}$ , changes in system performance are very small for FFT and RadixLocal as well. Both FFT and Radix exhibit similar performance for each parameter configuration for 0 and  $10^{-3}$  error rates. However, in each error rate performance varies up to 19%. Finally, at high error rates ( $10^{-3}$  or above), performance starts to deteriorate significantly (by more than 20%) when compared to the failure-free case.

## 6.2 Dynamic System Mapping

Table 3 shows the mapping time and the breakdown of the different types of probe messages exchanged in the event of a network topology change. In particular, a node is re-connected to a different location of the system and the first packet exchange triggers the mapping process. The different types of probe messages serve the purpose of locating hosts, locating switches, and distinguishing new switches from old ones, if necessary (i.e. unlike hosts, Myrinet switches do not have identities) [22]. The number of probe messages exchanged is linear with respect to the size of the network due to the breadth-first search. The number of probe messages, and thus the overall mapping time, may be further reduced [22], but we do not pursue this direction here.

## 7. CONCLUSIONS

System area networks have been the subject of extensive research, especially with respect to performance issues. However, little work has been done in the areas of fault tolerance and system availability. Our work is a first step towards this direction. We investigate how reliability can be incorporated in modern SANs and how they behave under errors. As such, this work is especially relevant for commercial SANs, which are now being deployed in applications with reliability demands, such as storage systems [33]. We demonstrate how reliability schemes can be incorporated in modern NICs and we investigate the performance implications.

We use a reliability protocol implemented in the NIC that takes advantages of sender-side buffering to tolerate transient network failures. We also propose an extension to existing network mapping schemes for dealing with permanent network failures transparently at the NIC level. Our scheme performs on-demand mapping of system nodes and does not require full system mapping. It relies on the retransmission mechanism to deal with deadlocks, if they occur, and it does not require a central map manager.

We study the retransmission protocol behavior with re-

# Hops (i.e. Links)	Host	Switch	Total	Mapping Time (ms)
1	28	0	28	3.054
2	53	20	73	25.855
3	83	41	124	48.488
4	113	73	186	83.567

**Table 3:** Dynamic routing performance summary. The columns labeled *Host* and *Switch* refer to host and switch probe messages respectively.

spect to various system parameters and determine a set of values for these parameters that offer fast recovery time in the presence of high network errors while limiting the overhead in the absence of network errors. Our retransmission protocol imposes a 20% overhead on system latency for small messages in the failure-free case, increasing latency from 8 to 10  $\mu s$ . The overhead imposed on bandwidth is less than 4% for all message sizes above 4 KBytes. The parameter space exploration for our retransmission scheme shows that it is very robust with respect to error rate. System performance, both with micro-benchmarks and real applications, starts to degrade significantly only when error rates are higher than  $10^{-3}$ . Finally, the retransmission scheme is rather insensitive to the network send queue size and even relatively small numbers of buffers on the send side result in little or no performance degradation.

## 8. ACKNOWLEDGMENTS

The authors would like to thank Peter Jamieson and Eugenia Distefano for helpful hints on setting up and using the experimental testbed. We would like to thank the members of the ATHLOS project for the useful discussions during the course of this work. Also, we thankfully acknowledge the support of Natural Sciences and Engineering Research Council of Canada, Canada Foundation for Innovation, Ontario Innovation Trust, the Nortel Institute of Technology, and Communications and Information Technology Ontario.

## 9. REFERENCES

- [1] T. Agerwala, J. L. Martin, J. H. Mirza, D. C. Sadler, D. M. Dias, and M. Snir. SP2 system architecture. *IBM Systems Journal*, 34(2):152–184, 1995.
- [2] W. E. Baker, R. W. Horst, D. P. Sonnier, and W. J. Watson. A flexible servernet-based fault-tolerant architecture. In *Proc. of the 25th International Symposium on Fault-Tolerant Computing*, Pasadena, CA, 1995.

- [3] R. A. F. Bhoedjang, T. Ruhl, and H. E. Bal. Lfc: A communication substrate for myrinet. In *Fourth Annual Conference of the Advanced School for Computing and Imaging*, Lommel, Belgium, June 1998.
- [4] R. A. F. Bhoedjang, K. Verstoep, T. Ruhl, H. E. Bal, and R. Hofman. Evaluating design alternatives for reliable communication on high-speed networks. In *Proc. of The 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS9)*, Cambridge, MA, Nov. 2000.
- [5] A. Bilas, C. Liao, and J. P. Singh. Using network interface support to avoid asynchronous protocol processing in shared virtual memory systems. In *Proc. of the 26th International Symposium on Computer Architecture (ISCA26)*, May 1999.
- [6] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. Su. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 15(1):29–36, Feb. 1995.
- [7] R. Christodouloupoulou and A. Bilas. Dynamic data replication for tolerating single node failures in shared virtual memory clusters of workstations. In *Proc. of The Workshop on Caching, Coherence and Consistency (WC3 2001)*, June 2001.
- [8] B. N. Chun, A. M. Mainwaring, and D. E. Culler. Virtual network transport protocols for myrinet. In *Proc. of The 1997 IEEE Symposium on High Performance Interconnects (HOT Interconnects V)*, August 1997.
- [9] Cray. Cray t3e data sheets. <http://www.cray.com/products/systems/index.html>, 2001.
- [10] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, 36(5):547–553, 1987.
- [11] C. Dubnicki, A. Bilas, Y. Chen, S. Damianakis, and K. Li. VMMC-2: efficient support for reliable, connection-oriented communication. In *Proc. of The 1997 IEEE Symposium on High Performance Interconnects (HOT Interconnects V)*, Aug. 1997. A short version of this appears in *IEEE Micro*, Jan/Feb, 1998.
- [12] D. Dunning and G. Regnier. The Virtual Interface Architecture. In *Proc. of The 1997 IEEE Symposium on High Performance Interconnects (HOT Interconnects V)*, Aug. 1997.
- [13] J. Flich, M. P. Malumbres, P. Lopez, and J. Duato. Improving routing performance in myrinet networks. In *Proc. of the 14th International Parallel and Distributed Processing Symposium (IPDPS'00)*, Cancun, Mexico, 2000.
- [14] M. Galle. Scalable pipelined interconnect for distributed endpoint routing: The SGI SPIDER chip. In *Proc. of The 1996 IEEE Symposium on High Performance Interconnects (HOT Interconnects IV)*, 1996.
- [15] P. Geoffray, L. Prylli, and B. Tourancheau. BIP-SMP: High performance message passing over a cluster of commodity SMPs. In *Proc. of The 1999 Supercomputing Conference on High Performance Networking and Computing (SC98)*. IEEE, Nov. 1999.
- [16] R. Gillett, M. Collins, and D. Pimm. Overview of network memory channel for PCI. In *Proc. of the IEEE Spring COMPCON '96*, Feb. 1996.
- [17] H. Gregory, J. Thomas, P. McMahon, A. Skjellum, and N. Doss. Design of the BDM family of myrinet control programs, 1998.
- [18] InfiniBand Trade Association. Infiniband architecture specification, version 1.0. <http://www.infinibandta.org>, Oct. 2000.
- [19] D. Jiang, H. Shan, and J. P. Singh. Application restructuring and performance portability on shared virtual memory and hardware-coherent multiprocessors. In *Proc. of The 1997 ACM Symposium on Principles and Practice of Parallel Programming (PPoPP97)*, pages 217–229, Las Vegas, NV, 1997.
- [20] V. Karamcheti and A. A. Chien. Software overhead in messaging layers: where does the time go? *ACM SIGPLAN Notices*, 29(11):51–60, Nov. 1994.
- [21] J. Laudon and D. Lenoski. The SGI origin: A ccNUMA highly scalable server. In *Proc. of the 24th International Symposium on Computer Architecture (ISCA24)*, volume 25 of *ACM SIGARCH Computer Architecture News*, pages 241–251. ACM Press, 1997.
- [22] A. M. Mainwaring, B. N. Chun, S. Schleimer, and D. S. Wilkerson. System area network mapping. In *Proc. of the 7th Annual ACM SIGPLAN Symposium on Parallel Algorithms and Architectures (SPAA97)*, pages 116–126, Newport, Rhode Island, June 22–25, 1997. SIGACT/SIGARCH and EATCS.
- [23] J. C. Martinez, F. Silla, P. Lopez, and J. Duato. On the influence of the selection function on the performance of networks of workstations. In *Proc. of the 3rd International Symposium on High Performance Computing (ISHPC 2000)*, pages 292–299, Tokyo, Japan, 2000.
- [24] J. Molero, F. Silla, V. Santonja, and J. Duato. Performance sensitivity of routing algorithms to failures in networks of workstations. In *Proc. of the 3rd International Symposium on High Performance Computing (ISHPC 2000)*, pages 230–242, Tokyo, Japan, 2000.
- [25] X. Molero, F. Silla, V. Santonja, and J. Duato. On the effect of link failures in fibre channel storage area networks. In *Proc. of the 2000 International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN 2000)*, Dallas, Texas, 2000.
- [26] 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.
- [27] S. Pakin, V. Karamcheti, and A. A. Chien. Fast Messages: Efficient, portable communication for workstation clusters and massively parallel processors (MPP). *IEEE Concurrency*, 5(2):60–73, April-June 1997. University of Illinois.
- [28] M. D. Schroeder, A. D. Birrell, M. Burrows, H. Murray, R. M. Needh, T. L. Rodeheffer, E. H. Satterthwaite, and C. P. Thacker. Autonet: a high-speed, self-configuring local area network using point-to-point links. Technical Report 59, Digital Equipment Corporation, Systems Research Center, Palo Alto, CA, Apr. 1990.
- [29] M. D. Schroeder, A. D. Birrell, M. Burrows, H. Murray, R. M. Needham, T. L. Rodeheffer, E. H. Satterthwaite, and C. P. Thacker. Autonet: a high-speed, self-configuring local area network using point-to-point links. Technical Report 59, Digital Equipment Corporation, Systems Research Center, Palo Alto, CA, Apr. 1990.
- [30] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, Inc., Upper Saddle River, NJ, 3rd edition, 1996.
- [31] H. Tezuka, A. Hori, and Y. Ishikawa. PM: A high-performance communication library for multi-user parallel environments. Technical Report TR-96015, Real World Computing Partnership, Nov. 1996.
- [32] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-net: a user-level network interface for parallel and distributed computing. In *Proc. of the Fifteenth Symposium on Operating Systems Principles (SOSP15)*, pages 40–53, Dec. 1995.
- [33] Y. Zhou, A. Bilas, S. Jagannathan, C. Dubnicki, J. Philbin, and K. Li. Experiences with vi communication for database storage. In *Proc. of the 29th International Symposium on Computer Architecture (ISCA29)*, May 2002.