Dependable user-level socket over dual networks

Nader Mohamed\textsuperscript{a,}\textsuperscript{*}, Jameela Al-Jaroodi\textsuperscript{a}, Hong Jiang\textsuperscript{b}

\textsuperscript{a}Dependable Systems Middleware (DeSyM) Research Laboratory, Department of Electrical and Computer Engineering, Stevens Institute of Technology, Hoboken, NJ 07030, USA
\textsuperscript{b}Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, NE 68588-0115, USA

Available online 23 June 2005

Abstract

Message duplication over multiple links can enhance the communication reliability and availability among distributed processes running on clusters or networked workstations. In addition, message striping over multiple links can enhance communication throughput and transfer times. This paper introduces optimized techniques to provide a dependable user-level socket that enhances the reliability and performance of message deliveries over dual network interfaces. The proposed techniques are designed to work on top of reliable transport protocols. In addition, a technique for efficient duplication of messages on dual networks to enhance both communication reliability and performance on top of reliable transport protocol is proposed. A prototype socket based on this model is developed and evaluated. The experimental evaluations of the proposed model indicate good communication reliability and performance gains over conventional methods.

Keywords: Dependable network services; Socket; Cluster networks; Reliable transport protocols

1. Introduction

Clusters \cite{5} are becoming increasingly widespread and cost-effective platforms for resource intensive applications such as multimedia, visualization, process controls, and scientific simulations. Most of these applications are time- and reliability-sensitive applications. One of the main requirements of successfully running these applications on a cluster is a dependable communication infrastructure among cluster nodes. Most clusters are equipped with dual networks that connect all nodes. These dual networks can be utilized to enhance the performance and dependability of communications in the cluster. However, there are many distributed applications running on clusters using TCP/IP protocols, thus requiring a fixed IP address to identify a machine. This leads to the necessary requirement that a given application use a single IP address on each node to establish connections with other nodes, thus preventing the application from utilizing the second available network interface to enhance the dependability of the communications. This implies that the communication capability and availability is bounded by the capability and availability of the single network associated with the assigned IP address. As a result, even if there are multiple networks connecting the nodes of the cluster, only one of them will be utilized for a given application.

There have been some efforts made to utilize multiple networks to enhance the communication performance. One important approach is the striping technique \cite{4}, exemplified by the Multiple-Network-Interface Socket (MuniSocket) \cite{10}. MuniSocket is a middleware level solution that enhances the performance of large messages on top of multiple networks. Another example is channel-bonding \cite{3} which is designed to increase communication throughput in Linux clusters. A third example is the sun IP network multipathing IPMP \cite{16} that enhances the throughput and reliability of outbound traffic. One main advantage of MuniSocket is that it is designed and implemented to be a network service, which works on...
top of any transport protocol without the need for specialized kernel support. As a result, and unlike channel-bonding and IPMP, MuniSocket works on heterogeneous networks and heterogeneous systems. As a network service, MuniSocket can also support cluster applications such as visualization and parallel file systems that need high-bandwidth and high-throughput communication for large messages. In this paper, we introduce some techniques to enhance the communication performance and reliability of MuniSocket. These techniques rely on the existence of a reliable transport protocol such as TCP. The techniques devise efficient duplication of messages on dual networks to enhance the communication reliability as well as performance.

In the rest of this paper, we first start with a discussion on background information and related work on network fault tolerance, striping, and MuniSocket, in Section 2. In addition, we identify the missing features of MuniSocket in terms of communication reliability. Section 3 evaluates the current MuniSocket overheads. Then, in Section 4, we introduce some techniques to reduce the overheads and enhance both the communication reliability and performance of MuniSocket. Section 5 evaluates the proposed enhancements. Section 6 concludes the paper.

2. Background and related work

In this section, we discuss some background information and related work related to network fault tolerance and striping for networks in clusters. In addition, we identify the missing features of MuniSocket in terms of communication dependability.

2.1. Network fault tolerance in clusters

Network fault tolerance is one of the main requirements for any reliable cluster system. Various techniques for network fault tolerance in clusters have been investigated. Examples of these techniques include Low Overhead Fault Tolerant Networking in Myrinet [9], Fault-Tolerant Ethernet Middleware (FTE) [17], and the Solaris IP Multipathing (IPMP) [16]. The Low Overhead Fault Tolerant Networking technique was developed to recover from network interface failures due to network processor hangs, where fault recovery is achieved by restoring the state of the Myrinet NIC using a backup copy.

FTE was designed to recover from failures of network interfaces, cables, and switches by utilizing network redundancy. It uses channel diagnostic messaging in which each node listens for a set of diagnostic messages on its channels from other nodes. A missing message from one channel indicates a potential failure, which results in switching to an alternate network to continue the communication. This technique minimizes the network fault detection time. Although FTE requires an extra network to enhance network reliability, it fails to utilize the extra communication bandwidth available in the alternate network. In other words, the communication bandwidth between any two nodes is bounded by the bandwidth of a single NIC or single network.

IPMP provides network fault tolerance by detecting the failure or repair of a NIC and switching the network address to and from the alternative NIC. IPMP detects a NIC failure and automatically switches network access to an alternate NIC. In addition, IPMP spreads outbound network packets across multiple NICs in order to achieve higher throughput. Although IPMP enhances outbound bandwidth, it does not enhance inbound bandwidth. This implies that for a destination node with multiple NICs only one such NIC will be utilized. Thus, the inbound bandwidth will be bounded by the bandwidth of a single NIC.

All of the above-mentioned techniques use network fault detection and recovery approaches to enhance network reliability. However, network failure detection and recovery processes are known to suffer severely from slow execution [17]. For example, it takes slightly less than 2 s for the Low Overhead Fault Tolerant Networking in Myrinet to detect and recover from a local NIC fault [9] and it takes FTE 1 to 2 s to detect and recover from a network fault. During the detection and recovery times, messages cannot be transferred, implying a high impact of a fault and fault tolerance on the small and medium size messages. For example, while a small message of size 32 Bytes takes around 0.05 ms to transfer from one node to another in Ethernet switched clusters, it takes more than 1 s if a network fault occurs in the above-mentioned techniques. This is because the network fault detection and recovery times become dominant in the message transfer time.

2.2. Striping in networks

With the advent of clusters with dual interconnects among nodes, some efforts have been made to utilize existing dual networks in these clusters to enhance the communication performance among the nodes. One important approach to provide such utilization is the striping technique [4], which is well known in with the context of storage systems such as the redundant arrays of inexpensive disk (RAID) architecture [8]. However, in networks, striping is used to describe the aggregation of multiple networks to achieve higher bandwidth and hence higher throughput. Striping may be implemented at different network layers, such as the physical, data link, network, transport, and applications layers. One example of striping in networks is Channel bonding [3]. Channel bonding uses multiple Ethernet networks and a striping at the data link (Ethernet) layer, which increases communication throughput but does not provide fault tolerance. Some work [2,7,18] has been done to minimize the overhead of striping at lower network layers by using queues at the receiving ends of channels to maintain synchronization between sender and receiver to reorder the packets. These methods also enhance throughput but do not provide fault tolerance.
Striping can be implemented as network services for applications requiring high network bandwidth. These network services are usually implemented at the middleware level between the transport protocols and the applications. One example of middleware solutions is MuniSocket [10,13], a user-level (middleware) socket model that utilizes multiple physical network connections on a cluster [11] or among grid components in wide-area-networks (WAN) [12] transparently from the application. In our previous work in MuniSocket we enhanced transfer performance of medium and large messages using TCP [11] and UDP [10] on multiple networks. We also enhanced the bulk data transfer scalability in WAN [12].

Striping at the socket and transport layers has been investigated by many research projects, such as the multiple TCP connections [14], the multiple streams [6], and the parallel socket [15]. However, these studies focus solely on achieving optimal bandwidth by overcoming the limitations imposed by the TCP window size on a single physical network connection, by means of parallel logical flows. In other words, such methods are used on networks with high delay*bandwidth product, such as WAN. In addition, these approaches are not designed to improve reliability or provide network fault tolerance.

2.3. MuniSocket

MuniSocket is designed to work on heterogeneous systems and on commodity-off-the-shelf (COTS) networks. It works on any reliable or unreliable transport protocol to provide an extensible communication bandwidth solution without requiring changes in the hardware or the kernel. MuniSocket is part of our efforts to create a middleware infrastructure for parallel and distributed programming models in heterogeneous systems [1].

In the MuniSocket model, user messages are fragmented into UDP datagrams or TCP packets where each datagram or packet has a unique sequence number and transferred in parallel, through multiple network interfaces, via one or more physical networks, to the destination, again through multiple network interfaces. At the destination, packets are reconstructed back to the original message. To have a reliable transfer service with the UDP-based MuniSocket, we devised reliability and congestion control mechanisms in MuniSocket itself [13].

MuniSocket is multithreaded, which includes one sending and one receiving thread per NIC, and a counter. MuniSocket divides large messages generated by the application into fragments that are then transmitted using the underlying transport protocol. Multiple concurrent threads are used to process the message and prepare the fragments for transmission. A sending thread is responsible for handling the transmission of some fragments on a specific NIC, while the receiving thread is responsible for handling received fragments and placing them into the appropriate place in a receiving buffer provided by the user. The counter provides a sequence number that identifies a fragment in the message. If the fragment size is \( F \) bytes and the message size is \( M \) bytes, then the counter provides sequence numbers from 0 to \( \lceil M/F \rceil - 1 \). Each sending thread tries to take a number, \( i \), from the counter, and sends both the sequence number \( i \), as a header, and the user data from location \( F \times i \) to \( F \times (i + 1) - 1 \) in the sender memory to the corresponding receiving thread in the receiving end over the corresponding network interface and network.

Implementing MuniSocket on top of a reliable transport protocol relieves MuniSocket from handling many end-to-end issues such as delivery guarantees, and flow and congestion control of the transferred message fragments. Both flow and congestion controls of the reliable transport protocols are utilized to implement the load-balancing algorithms among the sending threads. Load balancing is achieved by having threads connected to unloaded networks process more fragments while other threads connected to loaded networks are blocked for longer periods by the congestion and flow control mechanisms of the underlying reliable transport service [11].

MuniSocket increases bandwidth and throughput by fully utilizing multiple networks; however, the overall load on the networks may be unbalanced. In this case one network can become slower than the other and cause an overall delay in delivering the complete message. This imbalanced load can occur if the multiple networks used for transferring user messages of one application are shared with other applications. To illustrate this problem, assume that we are sending a 6 KB message using MuniSocket with three 2 KB fragments on two networks (see Fig. 1). The first sending thread sends the first fragment \( f_1 \) using the first network while the second sending thread sends the second fragment \( f_2 \) using the second network. After processing the first fragment, the first sending thread starts to send the third fragment \( f_3 \). At the same time another application with high network load starts on the first network. The time to send \( f_3 \) will increase significantly, as shown in Fig. 1. The impact of this problem is more pronounced in medium sized messages. For example, while sending a 6 KB message takes 2 time units on two unloaded networks, it might take 4 units if one of the networks is heavily loaded.

Thus, one of the problems associated with the current implementation of MuniSocket is inefficient utilization of shared heterogeneous resources with dynamic load. In [13] we discussed the basic technique to alleviate this problem only for large messages by designing a dynamic load-balancing algorithm that adapts itself to dynamic load changes. However, that basic technique does not cover cases with small to medium sized messages and does not address the issues of reliable transfer when faults occur. The main contribution of this paper, therefore, is the development of optimized and efficient techniques that enhance reliability and performance for small, medium, and large messages on dual networks. The proposed methods use efficient data du-
Fig. 1. Gantt chart of MuniSocket performance on unloaded networks (Case 1 on the left) and on loaded networks (Case 2 on the right).

3. MuniSocket performance

In this section we will discuss the performance of the original MuniSocket model to show where the overheads occur and to identify MuniSocket's limitations. Based on this evaluation, we will introduce our enhanced model (in Section 4) that overcomes these limitations, reduces the overheads and at the same time adds reliability to the original MuniSocket. To evaluate the performance gains of MuniSocket, a number of experiments were conducted on Sandhills, a 24-node cluster, where each node contains two AthlonMP 1.4 GHz processors with 256 KB cache and 1 GB RAM. The nodes were equipped with two fast Ethernet networks and a Gigabit Ethernet. The experiments measure the round trip time (RTT) and the effective bandwidth for the single network Socket and for MuniSocket using two fast Ethernet cards, with the effective bandwidth being derived indirectly from the RTT as follows:

\[
\text{Effective Bandwidth (Mbps)} = \frac{8 \times \text{Message size} \times 10^6}{\text{RTT}/2}.
\]

The communication performance in terms of RTT for TCP over fast Ethernet (TCP100) and MuniSocket over two fast Ethernet cards (MuniSocket 2 × 100) is shown in Fig. 2. The results show high RTT values for MuniSocket 2 × 100, as compared to TCP100, for messages smaller than 32 KB. However, as the message size increases the gain (reduction in RTT) for MuniSocket 2 × 100 becomes evident and clearly shows the benefits of this method. Fig. 3 shows the bandwidth gain incurred by MuniSocket on two fast Ethernet networks. The effective bandwidth reaches 186.4 Mbps for a 4 MB message. In [11], we have compared the performance of MuniSocket with channel bonding, where MuniSocket had slightly better performance and more flexibility.

The performance of MuniSocket is dependent on two factors. The first is the efficiency of the protocol and the second is the performance of the system resources such as processor speed, memory bandwidth, I/O bus bandwidth, and network interface and link bandwidths. While the protocol is designed to fully utilize these resources, it should also have reasonable overhead in message transfers. The overhead is generated from preparing fragment headers and thread contention on the shared counter, which is synchronized to maintain consistency, and correct fragment copying. To measure the actual utilization of MuniSocket from available resources, a modified version of MuniSocket is programmed to transfer the messages by logically partitioning the messages into equal parts and transfer them over two unloaded fast Ethernet interfaces. The modified version does not do any copying, physical partitioning, or adding headers. All sending and receiving threads work independently and there is no need for a fragment counter. This modified version measures the maximum achievable performance (Peak 2 × 100) on the available resources; the result is shown in Fig. 4.
Therefore, the existing fault tolerance mechanism is only
in the time needed to recover from faults. The next section
introduces techniques to eliminate MuniSocket’s limi-
tations and enhance the communication reliability on dual
networks, while at the same time enhance the performance.

4. Dependable MuniSocket over dual network interfaces

In this section, we propose some techniques to enhance
both the communication reliability and performance of Mu-
niSocket. The first is an efficient data duplication technique
to enhance reliability, load balancing, and overall perfor-
mance. The second technique relies on the functions of the
reliable transport protocol to eliminate some of the current
MuniSocket striping overheads. The third focuses on reduc-
ing the amount of duplication needed to enhance communica-
tion reliability and performance.

4.1. Enhancing communication reliability and performance

In order to enhance communication reliability and perfor-
mance through efficient message duplication in clusters with
dynamic load, the order of processing the fragments needs to
be changed. The shared counter of MuniSocket provides se-
quence numbers for the sending threads starting from 0 to the
number fragments in the message. If a sender processes and
sends a message that is divided into 8 fragments over dual
networks with dynamic load changes, the fragment distribu-
tion can be of different combinations. For example, the first
sending thread processes fragments \( f_0, f_3, f_4, \) and \( f_6 \), while
the second sending thread processes fragments \( f_1, f_2, f_5, \) and \( f_7 \). Fragments are sent with their sequence numbers con-
tained in their fragment headers. Receiving threads process
the header to get the fragment sequence number to copy the
fragments to the corresponding receiver buffer position.

Since striping is performed over reliable transport proto-
cols, the fragments’ first-in-first-out (FIFO) order is main-
tained for each sending and receiving thread pair. That is,
if sending thread \( i \) on the first machine sends fragments
\( f_0, f_3, f_4, \) and \( f_6 \) to the receiving thread \( i \) on the second ma-
chine, they will be received in the same order of \( f_0, f_3, f_4, \) and \( f_6 \). Although some fragments might be lost or corrupted,
underlying reliable transport protocol mechanism, such asTCP, maintains the reliability and FIFO order of fragment
delivery. The reliability mechanism ensures that there is no
packet loss or corruption.

The first step to enhance communication reliability is to
change the fragment processing order. With dual networks,
for example, there are two sending threads at the sender side
and two receiving threads at the receiver side. Each sending
and receiving thread pair is associated with a network. The
first sending thread can be set to send fragments from left to
right in the message buffer while the second sending thread
works from right to left, as illustrated in Fig. 6. The receiver
sends completion acknowledgment to the sender as soon
as it receives all fragments. The completion acknowledg-
ment is an indicator for the sender to stop processing more
fragments. For example, with \( n \) fragments, the first thread
sends fragments \( f_0, f_1, f_2, \ldots, f_m, f_{m+1}, f_{m+2}, \ldots, f_x \)
in sequence and the second sending thread sends fragments
\( f_{n-1}, f_{n-2}, f_{n-3}, \ldots, f_{m+1}, f_m, f_{m-1}, \ldots, f_y \) in

Fig. 4. Peak effective bandwidth using two fast Ethernet networks.

Fig. 5. Normalized bandwidth utilization of basic MuniSocket.
sequence, where the value of \( x \) is greater or equal to the value of \( y \). The proposed technique ensures that the network with higher bandwidth processes more fragments, while the network with lower bandwidth processes fewer fragments. The values of \( x \) and \( y \) depend on the load and available bandwidth on both networks. Therefore, this method inherently solves the problem of load balancing between dual networks. In addition, if one of the networks fails during sending the fragments, the proposed method inherently recovers from the faults by sending the rest of the fragments on the second network. The duplicated fragments are used to enhance the transfer time since the receiver will consider the fragments received first. If the last fragments sent on the first network are delayed due to other load on the network the receiver will receive the same fragments from the second network. If the message is small, the whole message is encapsulated in a single fragment that is duplicated on both networks and the receiver will consider the copy that arrives first.

The sender continuously sends fragments on each network as long as the receiver buffer can accommodate new fragments. The completion acknowledgement is only sent once when the receiver accepts all the fragments of the message. As a result, the maximum data duplication in unloaded networks (assuming large buffers) with bandwidth \( B \) and \( RTT \) can be estimated as \( 2 \times B \times RTT \). At the time that the receiver receives all fragments there are \( B \times RTT/2 \) bytes on their way from the sender to the receiver propagating on each network. In addition, another \( B \times RTT/2 \) bytes will be sent from the sender to the receiver on each network during the time needed for the completion acknowledgement to be transferred from the receiver to the sender. This is an estimated amount; however, the exact amount of fragment duplication depends on the sender and receiver buffer sizes. For example if the buffer is much smaller than the \( B \times RTT/2 \) bytes for each receiving thread, the duplication will also be small since transmission will be suspended for some time during the time from which the acknowledgement is sent by the receiver until it reaches the sender.

4.2. Striping overhead reductions

The reliability and FIFO properties of a reliable transport protocol such as TCP can be utilized to eliminate the need of sending sequence numbers with the fragments. The technique can be implemented by making each sending and receiving thread pair maintain two synchronized fragment counters. These counters are virtually duplicated one at the sending thread and one at the receiving thread. The counters can be synchronized by agreeing on the mode of the counters (counterMode), “increment” or “decrement”. When counterMode is “increment” the counter changes in increasing order (i.e., from left to right on the message buffer), and if it is “decrement” the counter changes in decreasing order (i.e., from right to left on the message buffer). The sender sends the size of the message to the receiver before sending the message fragments. Using the message and fragment sizes, the receiver calculates the number of fragments in the message. Using these values the sending and receiving thread pairs synchronize their fragment counters without using sequence numbers, thus eliminating the need for the fragment headers.

Using the number of fragments, the sending and receiving thread counters of the first network are initialized to zero with “increment” mode while the sending and receiving thread counters of the second network are initialized to the number of last fragment (lastFragment) with “decrement” mode. The sending and receiving threads of the first network know that they need to process fragments in the order of \( 0, 1, 2, 3 \), etc., without sending the fragment sequence numbers. In addition the sending and receiving threads of the second network process fragments in the order of \( f_{\text{lastFragment}}, f_{\text{lastFragment}}-1, f_{\text{lastFragment}}-2, f_{\text{lastFragment}}-3 \), etc. also without sending the fragments sequence numbers. The sending threads send the fragments one after the other and increase/decrease their counters until all fragments are sent.

The technique described above allows each pair of sending and receiving threads to maintain separate counters, where the receiving thread simulates the counter operation of its corresponding sending thread. Therefore there is no need to have a shared counter among the sending threads. Originally, using the shared counter among sending threads to provide fragments sequence numbers has a high possibility of contention because the counter becomes a single point of synchronization between the sending threads. Maintaining separate counters for each pair, as described above, eliminates the problem of contention.

4.3. Reducing the amount of data duplication

Based on our estimations in Section 4.1, the amount of data that is duplicated using the proposed technique for transmitting fragments is \( 2 \times B \times RTT \). This can be reduced to \( B \times RTT \) if an advanced completion acknowledgement is used. The advanced completion acknowledgement is sent by the receiver to the sender as soon as \( M-B \times RTT \) bytes are received by the receiver, where \( M \) is the size of the message. The advanced completion acknowledgement contains the numbers of the last fragments received from both networks. As soon as the sender receives this advanced completion acknowledgement, it knows the fragments that were correctly received and when each sending thread should
stop processing more fragments. In this case the estimated amount of duplicated data is reduced to $B \times RTT$ which is equivalent to the amount of data that can be sent on both networks during the transfer of the advanced completion acknowledgment from the receiver to the sender, $RTT/2$.

5. Analysis and performance measurements

This section evaluates the proposed techniques. The first subsection evaluates the enhancement of communication bandwidth and transfer time of medium and large messages. The second subsection evaluates the positive effects of message duplications in the transfer time for small messages. The third subsection evaluates the enhancement of the communication availability and the impact of this enhancement on message transfer times when a network fault occurs.

5.1. Enhancing communication bandwidth

The proposed method enhances the available bandwidth for small and large messages. If each network has random load while the average available bandwidth on one network is $b$, the average available bandwidth in both networks is $2 \times b$ when both networks are used simultaneously to transfer message fragments. Experiments were conducted to evaluate the dependable MuniSocket performance over two fast Ethernet networks. With the proposed mechanism, the overall performance improved noticeably (see Fig. 7). From the experimental results we see the effect of removing the fragment header, eliminating thread contention on the shared fragment counter, and using efficient data duplication. While MuniSocket (MuniSocket $2 \times 100$) only achieved speedup with messages of size 32 KB and larger, the dependable MuniSocket achieved speedup for messages of size 8 KB and beyond.

The speedup of dependable MuniSocket for a 64 KB message is around 1.86 and 1.47 relative to TCP100 and MuniSocket, respectively. The speedup of dependable MuniSocket for a 2 MB message is around 1.98 and 1.02 relative to TCP100 and MuniSocket, respectively. This shows that the overhead is high for medium messages but is largely hidden for large messages. Fig. 8 shows the bandwidth utilization of MuniSocket and dependable MuniSocket, normalized to the maximum achievable bandwidth. The effective bandwidth of MuniSocket approaches maximum achievable peak with 0.5 MB messages, while dependable MuniSocket achieves the peak with messages of size 16 KB.

Another set of experiments was conducted to measure the performance of the load-balancing mechanism in dependable MuniSocket. Two fast Ethernet networks were used in these experiments. The first network has no load, while the second has some load from messages of size 32 KB being exchanged between the two nodes. Fig. 9 shows the average effective bandwidth achieved with the standard TCP socket, using the first network (Unloaded TCP100) and the second network (Loaded TCP100), respectively. In addition, it shows the average effective bandwidth achieved with dependable MuniSocket utilizing the available bandwidth in both networks. Dependable MuniSocket is shown to achieve performance speedups with messages of size around 40 KB and larger, and to demonstrate the dynamic load balancing ability. The results also show that while the loaded network provides less than 75% of its peak bandwidth, MuniSocket is still able to achieve high bandwidth gain with dynamic load balancing. In other words, with a 2 MB message, for example, an average bandwidth of 82.123 Mbps was obtained with either network interface when not using MuniSocket. However, using dependable MuniSocket, an average bandwidth of 161.84 Mbps was achieved using both interfaces simultaneously.

5.2. Enhancing transfer time for small messages

The proposed solution enhances the performance of transfer for small messages. Assume that multiple applications are running on a cluster with dual networks. Some applications use the first network while the rest use the second network. Thus, unrelated random loads are generated on both
networks. If each network has probability of having other load on one network is \( q \), where \( 0 \leq q \leq 1 \), the probability of delay in transferring small message because of the other load is also \( q \). The probability of having a load on both networks is \( q^2 \). Since important small messages are duplicated by dependable MuniSocket on both networks, then the probability of delaying the transfer of a small message in the networks because of other loads is also \( q^2 \). For example, if each network is used on average 20% by using standard socket, \( q \) will be equal to 0.2. The probability of delay in sending a small message because of other load in the network is also 20%. However, using the proposed solution, the probability of delaying the transfer of the small message because of other load in the networks reduces to \( q^2 \) which is 4%.

To show the benefits of duplicating small messages over two networks, experiments were conducted to measure the negative impact of transmitting large messages on small message latency. The experiments involved two machines where two pairs of processes exchange messages. The experiments were designed to measure the impact when both messages are sent and received through the same pair of network interfaces. The first pair of processes exchanges small 32 Byte messages, while the second pair exchanges large messages of predefined sizes. The experiment measures the average RTT of transferring the 32 Byte messages at the same time when messages of other sizes were being exchanged. While the average RTT for 32 Byte messages on fast Ethernet was 0.123 ms when there is no other load from the second pair, it increased 8.6 times to 1.055 ms with a load of 16 Kbytes from the second pair. In addition, the average RTT increased 35.8 times to 4.398 ms with a load of 1 Mbytes from the second pair. While some coordination and resource allocation may be done for messages going from and to the same source and destination nodes at the same time, it becomes difficult when there are other messages sent by other nodes in the cluster to the same destination node. When two networks are available using dependable MuniSocket, the small messages can be duplicated and sent concurrently on both networks. The receiver will take the first arriving message and discard the other. As a result the transfer time will be the shortest among the two networks thus if one of the networks has other load, the message on the other network will arrive first. In addition to enhancing the transfer time, this technique also improves the reliability of the transfer because even if one of the networks completely fails, the message will still arrive in a timely manner.

5.3. Reliability enhancement analysis

The proposed method enhances the reliability of transferring the messages by utilizing both networks. If the availability of each network is \( p \), where \( 0 \leq p \leq 1 \), then the fault probability of each network is \( (1 - p) \). Thus the probability of faults in both networks at the same time is \( (1 - p)^2 \). The probability of availability of at least one network is \( 1 - (1 - p)^2 \) which is the availability of the networks using the proposed method. We also have \( 1 - (1 - p)^2 > p \). For example, if we have two networks connecting the nodes in a cluster and the availability of each network is 97%. The availability of the communication using standard socket is also 97%. With the dependable socket, the availability of the communication is increased to 99.91%.

Experiments were conducted to evaluate the impact of a single network failure and the reliability mechanism on message transfer times. The experiments were designed to compare the performance of the proposed reliability mechanism with the detection and recovery approaches such as that proposed in FTE [17]. We selected FTE because it uses efficient and fast mechanisms for network fault detection and recovery. In addition, it uses an extra network to be used as a standby network. However, FTE only uses the standby network for data transfer when there is a fault in the active network in use. Both FTE and dependable MuniSocket require the same configuration environments and recover from the same types of network failures. The evaluation used in FTE was failover time which is the time elapsed between when a failure occurs in the active network and when the related nodes have switched from the faulty network to the standby network. The failover time is the total time it takes the FTE software to detect a failure and recover from it. FTE requires 1 to 2 s failover time [17].

The experiments were conducted using a discrete event simulator that performs a time-step simulation of network operations. The simulated environment consists of two nodes linked by dual fast Ethernet networks. The dual fast Ethernet networks linking two nodes. Before we start using the simulator for these experiments, we ensured that the performance of the fast Ethernet network in the simulator is similar to the performance of an actual fast Ethernet in a cluster environment with a single fast Ethernet network. In addition, we have ensured that the performance of using dual networks in the simulator is similar to the performance of using the actual dual networks with the actual implementation of MuniSocket.
These techniques utilize the reliability and FIFO properties when a network fault occurs.

Fig. 10. The impact of reliability mechanisms on message transfer times when a network fault occurs.

A fault in one of the networks was simulated. The fault occurs in one of the networks after sending half of the message on the two networks. Three sets of experiments were conducted. The first set is for the proposed approach in this paper. The second is for the detection and recovery approach in which the failover time is 1s. The third set is for the detection and recovery approach in which the failover time is 2s. The network fault detection and recovery used is similar to that used in FTE. The experiments measured the transfer times for different message sizes while a network failure is occurring after sending half of the message. The results are shown in Fig. 10. As observed, the impact of a network failure on the transfer time performance in the proposed approach is much less than the detection and recovery approach. This is because that the proposed approach inherently enhances the communication reliability on dual networks without using network fault detection and recovery mechanisms. Thus it will automatically transfer the remainder of the message on the available network without wasting failover time.

Although, the proposed approach involves sending duplicated fragments for each message sent through a dependable MuniSocket, the network fault detection and recovery involves periodic diagnostic messages to discover network faults. This will generate numerous additional messages. However, in MuniSocket only important messages can be sent with efficient message duplication to enhance communication reliability while all other messages can be sent using single or dual networks without any duplication. This means that only important messages will generate duplicated fragments.

6. Conclusion

We proposed techniques to enhance the dependability of the reliable-transport-based MuniSocket on dual networks. These techniques utilize the reliability and FIFO properties of the underlying reliable transport protocols to enhance reliability, transfer times, and to overcome some of the limitations associated with dynamic loads on the networks. The first technique deploys an efficient data duplication method to enhance communication reliability, load balancing, and overall performance. The second technique relies on utilizing the functions of the reliable transport protocol to eliminate some of MuniSocket’s striping overheads. The third focuses on reducing the amount of duplication needed to enhance communication reliability and performance. The proposed techniques were evaluated and the results illustrate good communication reliability and performance gains compared to the standard Socket and the original MuniSocket implementations. In addition, compared to many other approaches for enhancing reliability, MuniSocket with the introduced techniques has the additional advantage of enhancing the performance along with reliability.

In general, MuniSocket has been implemented to transparently utilize as many network interfaces as available. However, the reliability and performance enhancing techniques reported in this paper addressed the issue with only dual networks, which is the common case in clusters. We are currently considering the different possibilities to use these techniques with more than two networks. Furthermore, we plan to introduce additional techniques to optimize network utilization based on dynamic and real-time monitoring of the networks properties and appropriately selecting the most suitable technique for message transfer.

Acknowledgments

This project was partially supported by a National Science Foundation Grant (EPS-0091900), a Nebraska University Foundation Grant, and an Academic Priority Grant of the University of Nebraska-Lincoln. We would like to thank members of the secure distributed information (SDI) group and the research computing facility (RCF) at the University of Nebraska-Lincoln for their continuous support.

References


Nader Mohamed is an Assistant Professor of Electrical and Computer Engineering, Stevens Institute of Technology, New Jersey, USA. His main research interests include middleware, networking, cluster and Grid computing, systems integration, and dependable systems. He obtained a Ph.D. in Computer Science from University of Nebraska-Lincoln; 2004, an M.Sc. in Computer Science from Western Michigan University; 1998, and a B.Sc in Electrical Engineering from University of Bahrain; 1992. He has over 8 years of industry experience in systems integration and middleware.

Jameela Al-Jaroodi is a Research Assistant Professor at The Department of Electrical and Computer Engineering, Stevens Institute of Technology. Her research interest is in distributed systems including middleware, distributed software for heterogeneous systems, ad hoc networks, sensor networks, and software engineering for distributed systems. She published more than 30 articles in Journals and Conferences. Dr. Al-Jaroodi received her doctoral degree from The Computer Science and Engineering Department at The University of Nebraska-Lincoln in 2004; her master degree from Western Michigan University in 1998; and her bachelor degree from The University of Bahrain in 1993.

Hong Jiang received a B.Sc. in Computer Engineering from Huazhong University of Science and Technology, Wahan, China; 1982, an M.Sc. in Computer Engineering from the University of Toronto, Toronto, Canada; 1987, and a Ph.D. in Computer Science from Texas A&M University, College Station, Texas, USA; 1991. Since August 1991 he has been at the University of Nebraska-Lincoln, Lincoln, Nebraska, USA, where he is Professor and Vice Chair in the Department of Computer Science and Engineering. His present research interests are computer architecture, parallel/distributed computing, computer storage systems and parallel I/O, performance evaluation, networking, and computational engineering. He has over 80 publications in major journals and international conferences and his research has been supported by NSF, DoD and the State of Nebraska.