Self-Configuration Techniques for MuniSocket

Nader Mohamed and Jameela Al-Jaroodi
The College of Information Technology
United Arab Emirates University
Al Ain, P.O. Box 17555, UAE, {nader.m, j.aljaroodi}@uaeu.ac.ae

Abstract

MuniSocket (Multiple-Network-Interface Socket) provides mechanisms to enhance the communication performance properties such as throughput, transfer time, and reliability by utilizing the existing multiple network interface cards between a pair of computers. Although the MuniSocket model has some communication performance advantages over the regular socket, it has a number of configuration complexity drawbacks including the complexity of establishing multiple channels and configuration for good communication performance under different connection scenarios. This paper develops some self-configuration techniques for MuniSocket. These techniques are self-discovery technique for discovering the existence of network interfaces and their performance properties, self-configuration for establishing channels over the interfaces, and self-optimization for selecting the best channels combinations for sending various message sizes. While these techniques enhance communication performance among computers, they also reduce the complexity of configuring MuniSocket and making its interface compatible with the regular TCP socket interface.

Keywords: Socket, Multiple Network Interfaces, Self-Managed Networks.

1. Introduction

Computers in local area networks or wide area networks may be connected through multiple homogeneous or heterogeneous network interfaces such as wireless, fast Ethernet, and Gigabit Ethernet though one or more switches/routers. Utilizing these multiple interconnections among multiple computers to enhance communication performance and reliability is very complex especially with dynamic and heterogeneous environments. One approach to utilize the existing multiple network interfaces between a pair of nodes is the MuniSocket model [7]. MuniSocket provides mechanisms to enhance the communication performance properties such as throughput, transfer time, and reliability. MuniSocket is a user-level socket layer over multiple physical network interfaces. It provides mechanisms to simultaneously utilize multiple network interfaces among multiple computers in local area networks and wide area networks. In addition, MuniSocket has been used in a number of applications that require huge data transfer such as distributed file system. However, MuniSocket requires some manual configurations that require various measurements and setup from users to utilize the existing multiple network interfaces and networks. In this paper we propose to develop a self-managed MuniSocket that deals with multiple homogeneous and heterogeneous network interfaces and networks. The aim here is to make MuniSocket capable of managing itself to achieve enhanced communication performance under various interconnection scenarios.

Although MuniSocket provides some mechanisms to enhance the communication performance [7] and reliability [9], it requires several configuration steps from users including defining the multiple IP addresses and ports for connection establishment between the client and server nodes. In addition, MuniSocket does not configure itself for dealing with different communication message lengths for better optimization on heterogeneous network interface cards and links. This paper discusses some enhancements for MuniSocket by using self-configuration techniques that will relieve users from manually performing those configuration tasks.

In the rest of this paper, we first discuss some background information about utilizing multiple network interface cards, MuniSocket, and autonomic computing in Section 2. Section 3 provides the techniques for self configuration. Performance enhancements are discussed in Section 4. Section 5 concludes the paper and provides some remarks about future work.
2. Background

A lot of research is being invested in providing mechanisms to support and utilize multiple networks. Each one provides solutions for a certain property such as throughput and reliability. However, most of these solutions require some physical network configurations that enforce other limitations which make them inflexible. These limitations are mostly imposed by the low level implementations of the solutions. In addition, most of these solutions do not provide any self-configuration mechanisms which makes them work efficiently only for limited types of scenarios. One example of these solutions is Channel Bonding that is designed for Linux clusters connected with multiple fast Ethernet networks. Channel Bonding has some limitations such as its system dependence, MAC address and switch restrictions, inability to access individual bonded cards, and requiring that all nodes on the cluster be bonded and have the same type and number of interfaces. Another example is IPMP (Solaris IP Network Multipathing) from Sun Microsystems. IPMP is incorporated in Solaris 8 and 9, where the striping is done at the IP level to provide dynamic load balancing and fault tolerance on multiple network interfaces. One of the main limitations of IPMP is that it only enhances outbound communication bandwidth. Detailed study of these solutions, their limitations, other solutions, and their references can be found in [7][9].

MuniSocket was designed at the middleware level to overcome some of the limitations of other approaches.

Table 1. Different scenarios for using MuniSocket to utilize the available bandwidth.

<table>
<thead>
<tr>
<th>Case 1:</th>
<th>Two Computers connected through two homogeneous networks. MuniSocket can be used to utilize both networks.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case 2:</td>
<td>Two computers in a LAN connected through wired and wireless networks. MuniSocket can be used to utilize both networks.</td>
</tr>
<tr>
<td>Case 3:</td>
<td>A mobile notebook connected to a LAN through two wireless adapters to increase the available bandwidth. MuniSocket can be used to utilize both adapters.</td>
</tr>
<tr>
<td>Case 4:</td>
<td>Each computer has two or more interfaces connected to two or more separate dedicated links in WAN. MuniSocket with multiple channels can be used to avoid both interface and network limitations and to increase the effective bandwidth between the computers.</td>
</tr>
<tr>
<td>Case 5:</td>
<td>Left computer has two Fast Ethernet interfaces and right computer has a single Gigabit Ethernet interface. The machines are connected to LANs, which are connected to a WAN. MuniSocket can be used to avoid network interface limitation from left computer side and to increase effective bandwidth between the machines.</td>
</tr>
<tr>
<td>Case 6:</td>
<td>Two computers connected through heterogeneous number of interfaces through a shared WAN and a dedicated line.</td>
</tr>
</tbody>
</table>
For example, while it enhances communication throughput [7], it also enhances the communication reliability [9]. It can be used for different LAN and WAN interconnection scenarios, see Table 1. MuniSocket has been proposed to be used in a number of applications that require huge data transfer. Examples of these applications are a high throughput and scalable file system [13] and high-throughput communication for Grid computing [1][10].

Autonomic and self-management technologies [6][14] are receiving increasing attention for addressing usability issues of complex systems such as distributed systems, storage systems, and network systems, because as these systems have the ability to manage themselves with no or minimum user involvement, the utilization degree of the system will be significantly increased. Autonomic and self-management techniques were embedded in a number of network systems [5]. While most of the previous work concentrated on enhancing different networking services such as security and QoS, here in this project we concentrate on developing self-configuration techniques to efficiently utilize existing multiple network interfaces to enhance communication performance among multiple machines. In our previous work, we developed some self-configuration techniques for a specific environment that is for clusters with multiple nodes with multiple interconnections [8]. The aim here is to develop generic self-configuration techniques for any environment where MuniSocket can be used.

3. Self-Configuration Techniques

In this section multiple techniques for self-managed MuniSocket are proposed. Some of these techniques address channels discovery and configuration issues which are explained in Subsection 4.1. Subsection 4.2 explains the self-configuration technique that selects whether a message will be sent through a single or multiple homogeneous channels. Subsection 4.3 explains self-configuration for selecting the best channel that a small message could be transferred through when multiple heterogeneous channels are available.

3.1 Self-Discovery and Self-Configuration of MuniSocket Channels

In a regular TCP socket, the user needs to establish a connection between a client and a server before any messages can be transferred between them. In order for the user to establish the connection, he/she needs to know the server IP address and port number. The server IP address is the address related the network interface card he/she wishes to connect to in case there is a number of network interface cards in the server. The port number is the number used by the server to identify the associated process. Both the IP address and port number must be defined at the client side.

One of the main differences between MuniSocket and a regular TCP socket is that in MuniSocket users need to define multiple IP addresses and port numbers for establishing multiple logical channels between a client and a server. These multiple logical channels will be used by MuniSocket for distributing and transferring communication message fragments in case of parallel transfers [7]. For example, the user can define two logical channels between two nodes in which each node has two Fast Ethernet network interface cards and all interface cards are linked to a fast Ethernet switch as in Figure 1. In this network, each interface is defined by \((i, j)\) where \(i\) is node number and \(j\) is interface number in node \(i\). For the previous example, the two channels are from interface \((X,1)\) to interface \((Y,1)\) and from interface \((X,2)\) to interface \((Y,2)\).

For better configuration, the user can define four logical channels from node \(X\) to node \(Y\). The four channels are from \((X,1)\) to \((Y,1)\), from \((X,1)\) to \((Y,2)\), from \((X,2)\) to \((Y,1)\), and from \((X,2)\) to \((Y,2)\). The message fragments are distributed and transferred among these four channels. Although in this configuration physical links are shared among the channels, it has been proven that it has better communication performance than the pervious configuration [11].

Another example is shown in Figure 2. Two nodes \(N\) and \(M\) are connected through a Gigabit Ethernet switch. In the first node \(N\), there are two Fast Ethernet network interface cards while there is only one Gigabit
network interface card on node \( M \). Although theoretically there is 200 Mbps bandwidth between both nodes, a maximum of 100 Mbps can be achieved between both nodes if a regular TCP socket is used. With MuniSocket, a maximum of 200 Mbps can be achieved if the user defines two logical channels. The first channel is from \((N,1)\) to \((M,1)\). The second channel is from \((N,2)\) to \((M,1)\).

The process of defining multiple logical channels for MuniSocket makes the interface of MuniSocket different from the interface of a regular TCP socket. This unmatched interface makes them incompatible thus it becomes impossible to use MuniSocket instead of the regular socket directly. However, the process of establishing logical channels between two nodes can be achieved automatically without involving the users or changing the regular socket interface. This can be achieved by having self-discovery and self-configuring mechanisms as a part of MuniSocket.

The steps of the self-discovery and self-configuring mechanisms are as follows:
1. As soon as a server MuniSocket is started it will collect IP addresses of all network interface cards that are available in the server node before it listens to a specific port provided by the user.
2. As soon as a client is started, it is going to establish one connection with the server. This connection is based on the server’s published IP address and port number, which the user defines in the client MuniSocket. The user defines only a single reachable server IP address from the client and only one port number which is the server port number.
3. As soon as a connection is established. The server will send all of its IP addresses for all its interfaces to the client.
4. The client will receive the server IP addresses and find all of its IP addresses as well. The client will perform reachability analysis to see which interfaces in the client can reach which interfaces in the server. From this step the client can discover and establish the logical channels between the client MuniSocket and the server MuniSocket.
5. The client sends all possible logical channels to the server using the first connection.
6. The server prepares for establishing these logical channels and selects and sends a free port for each logical channel to the client.
7. As soon as the client receives the list of ports, it establishes all possible channels with the server and the communication is done through these channels.

The reachability analysis is achieved using echo messages. Based on the above steps, both regular TCP socket and MuniSocket will have the same interface. In both, at the server side, only a single server port is defined. In addition, at the client side, only a single server IP address and a single server port number are initially defined in both types of sockets. While keeping the same regular socket interface for MuniSocket, the proposed self-discovery and self-configuring mechanisms also relieve users from dealings with a list of IP addresses and port numbers and from having to manually perform the reachability analysis.

### 3.2 Self-Configuration for Data Transfer using Homogeneous Channels

MuniSocket provides an efficient mechanism to utilize multiple homogeneous interfaces to achieve higher bandwidth. However, this mechanism does not give better performance at all message sizes. Using fast Ethernet, the experiments show some overhead for messages smaller than 1KB, but after that the performance improves dramatically giving much better results than TCP Socket. To avoid such overhead with small messages, a dynamic decision has to be made when a message is received for transmission. The system should decide whether to divide the message or send it as is on a single interface based on the message size and a pre-determined cut-off value. To demonstrate this mechanism, assume we have two network interfaces. Let \( RTT_1 \) be the return trip time of a message \( M \) on a single network and \( RTT_2 \) be for two networks, which includes the striping overhead. The striping overhead is slightly dependent on the message size; therefore, the self-managed MuniSocket should calculate the message size at the intersection point where \( RTT_1(M) = RTT_2(M) \) (for example, the intersection point between single TCP Socket using single Faster Ethernet Interface (TCP 100) and MuniSocket using two Fast Ethernet Interfaces in Figure 3).

The intersection point represents the desired message size to transition from using a single network interface to utilizing two network interfaces. Assuming dedicated networks, the system registers transmission times for both a single interface and two interfaces at start time. Using these determined values the cut-off message size can be calculated using an experimental search method, which can be conducted at start time. A binary search algorithm can be used, which can be further optimized by estimating rough bounds for the search. With the cut-off message size known, the system is able to decide whether to send the message on a single interface (when \( M < \text{cut-off} \)) or use two interfaces (when \( M \geq \text{cut-off} \)).
Self-Channel-Selection among Multiple Heterogeneous Channels

Transfer times of small messages, can not be enhanced using parallel transfer. However, the existence of multiple heterogeneous channels between two nodes can be utilized to enhance the transfer time for small messages. Heterogeneous channels can be found among two nodes due to heterogeneous links, heterogeneous switches and routers, and heterogeneous network interface cards among them. Heterogeneous channels usually have different latencies and bandwidths. MuniSocket can transfer small messages through the channel with smallest latency. MuniSocket can discover the latencies of the available channels during the reachability analysis that is discussed in the previous section. Table 2 contains different scenarios for different channel types with their round trip transfer time for a message with length 32 bytes. These measurements are conducted using two laptops with Intel Pentium M processor 1.80 GHz and 512 MB of RAM. The benchmark used for this experiment was Ping Pong [3][4].

4. Performance Enhancements

Multiple sets of experiments were conducted using a prototype self-managed MuniSocket. For all these experiments the Ping Pong benchmark [3][4] was used. The first set of experiments is for testing the MuniSocket parallel transfer using the two types of MuniSocket channels configurations: one-to-one and permutation. The second set of experiments is for finding the effect of self selection technique for sending a message either through a single or multiple channels. The third experiment is for testing the self channel selection for small messages among multiple heterogeneous channels. These sets of experiments were selected to cover the techniques developed in this paper. However, more general experimental measurements of the MuniSocket performance can be found in [7] for local area networks, in [10] for bulk data transfers in wide area networks, and in [9] for dependability measurements.

Table 2. Examples: Round Trip Time for small message of length of 32B on different channels.

<table>
<thead>
<tr>
<th>#</th>
<th>First Computer Interface</th>
<th>Second Computer Interface</th>
<th>Network Structure</th>
<th>RTT (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bluetooth USB Adapter</td>
<td>Bluetooth USB Adapter</td>
<td>Direct Wireless Link</td>
<td>27.1</td>
</tr>
<tr>
<td>2</td>
<td>Wireless-G USB Adapter</td>
<td>Built-in Wireless LAN 1450 Dual Band</td>
<td>Fast Ethernet Switch and Wireless Switch &amp; Router</td>
<td>2.0</td>
</tr>
<tr>
<td>3</td>
<td>Built-in Wireless WLAN 1450 Dual Band</td>
<td>Built-in Wireless WLAN 1450 Dual Band</td>
<td>Wireless Switch</td>
<td>1.6</td>
</tr>
<tr>
<td>4</td>
<td>Wireless-G USB Adapter</td>
<td>Fast Ethernet USB Adapter</td>
<td>Fast Ethernet Switch and Wireless Switch &amp; Router</td>
<td>1.4</td>
</tr>
<tr>
<td>5</td>
<td>Wireless-G USB Adapter</td>
<td>Built-in Gigabit Ethernet</td>
<td>Fast Ethernet Switch and Wireless Switch &amp; Router</td>
<td>1.2</td>
</tr>
<tr>
<td>6</td>
<td>Built-in Wireless WLAN 1450 Dual Band</td>
<td>Fast Ethernet USB Adapter</td>
<td>Fast Ethernet Switch and Wireless Switch &amp; Router</td>
<td>1.2</td>
</tr>
<tr>
<td>7</td>
<td>Built-in Gigabit Ethernet</td>
<td>Built-in Wireless WLAN 1450 Dual Band</td>
<td>Fast Ethernet Switch and Wireless Switch &amp; Router</td>
<td>1.0</td>
</tr>
<tr>
<td>8</td>
<td>Fast Ethernet USB Adapter</td>
<td>Fast Ethernet USB Adapter</td>
<td>Fast Ethernet Switch</td>
<td>0.5</td>
</tr>
<tr>
<td>9</td>
<td>Built-in Gigabit Ethernet</td>
<td>Fast Ethernet USB Adapter</td>
<td>Gigabit Ethernet Switch</td>
<td>0.3</td>
</tr>
<tr>
<td>10</td>
<td>Built-in Gigabit Ethernet</td>
<td>Built-in Gigabit Ethernet</td>
<td>Gigabit Ethernet Switch</td>
<td>0.1</td>
</tr>
</tbody>
</table>
In the first set of experiments parallel transfer for large messages were tested using the two types of channels configurations: one-to-one and permutation. This set of experiments is conducted between two nodes connected by two network interfaces cards in each one and a fast Ethernet switch, as in Figure 3. Extra artificial inbound and outbound loads were created in the first NIC in the first node and in the second NIC in the second node. These extra loads simulate message exchange with other nodes on the network. The MuniSocket with one-to-one connection configuration was evaluated. The average effective bandwidth between both nodes was around 94Mbps. In this configuration only two logical channels were established. When the MuniSocket with permutation connection configuration was evaluated, an average of 140Mbps effective bandwidth was achieved under the same load conditions. In this configuration four logical channels were established. As we notice, the permutation connection configuration achieved better communication performance.

In the second set of experiments, the effects of using the self selection technique for sending a message either through a single or multiple channels was measured in two scenarios. The first scenario was between two nodes connected through two Fast Ethernet networks. The result of using the suggested technique is shown in Figure 4. The second scenario was between two nodes connected through two Gigabit Ethernet networks. The result of using the suggested technique is shown in Figure 5. In both scenarios, the proposed self selection technique in MuniSocket could enhance the transfer time for short messages compared to the regular MuniSocket.

The third experiment was conducted to test the self channel selection for 32-byte messages among multiple heterogeneous channels. The experiment was conducted between two nodes connected through three channels: 1, 3, and 7 of Table 2. Channel 7 was selected by MuniSocket as it has the smallest latency time.

5. Conclusion and Future Work

This paper discussed self-configuration approaches to enhance the usability, manageability, and performance of MuniSocket. These approaches are based on logical channel discoveries. The logical channel discovery includes reachability and latency analysis. One of the main advantages of self-configuration techniques is that they make MuniSocket interface compatible with the regular TCP socket interface. As a result, applications using regular TCP sockets can easily switch to use MuniSocket. The techniques explained in this paper are self-discovery and self-configuration of channels techniques, the self-configuration technique selects whether a message will be sent through a single or multiple homogeneous channels, and self-selection is for choosing the best channel that a small message could be transferred through among multiple available heterogeneous channels.

Our plan for the future is to study and enhance additional self-management approaches for MuniSocket such as self-optimization and self-healing. This includes enhanced resource discovery, enhanced performance optimization that could monitor the current load on the channels and redistribute it as necessary to achieve load-balancing, and enhanced fault discovery and recovery for the self-healing approach.
Acknowledgment

This work is supported by United Arab Emirates University Research Grant # 04-03-9-11/07.

References