PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks

H. Sivakumar
sharinat@eecs.uic.edu
University of Illinois at Chicago
Chicago, IL, USA

S. Bailey
sbailey@infoblox.com
Infoblox Inc.
Evanston, IL, USA

R. L. Grossman
grossman@uic.edu
University of Illinois at Chicago
Chicago, IL, USA
Magnify Inc.
Chicago, IL, USA

Abstract

Transmission Control Protocol (TCP) is used by various applications to achieve reliable data transfer. TCP was originally designed for unreliable networks. With the emergence of high-speed wide area networks various improvements have been applied to TCP to reduce latency and achieve improved bandwidth. The improvement is achieved by network tuning that involves system administrators, performance study and it takes a considerable amount of time. This paper introduces PSockets (Parallel Sockets), a library that achieves the equivalent performance without the extensive process of tuning the network to achieve the maximum performance. The basic idea behind PSockets is to exploit network striping. By network striping we mean striping partitioned data across several open sockets. We describe experimental studies using PSockets over the Abilene network. We show in particular that network striping using PSockets is effective for high performance data intensive computing applications using geographically distributed data.

1. INTRODUCTION

With the rapid advancements in networking technologies, high-speed networks such as Abilene and vBNS (very high speed backbone network service) are becoming more common. Although distributed data mining and data intensive computing applications have an urgent need for the throughput provided by these high performance networks, achieving the desired throughput in practice can sometimes be difficult due to the network tuning that is usually required to achieve optimal performance. The key focus of this paper is to help applications to achieve the best end-to-end performance (maximum throughput) over high-speed wide area networks without modifications to the existing network.

Most data applications need reliable data transfer and hence utilize the Transmission Control Protocol (TCP) [1]. TCP was originally designed for unreliable networks that required reliable data transfers between the source and destination machines. A common way to improve the bandwidth and reduce the latency while using TCP on reliable high-speed wide area networks is to tune the TCP window size appropriately [2, 7]. This involves the system administrator at both ends and involves a considerable amount of time, ranging from a few hours to a week, to achieve the best window size for the best performance.

[1]

In this paper, we introduce a C++ library called PSockets (Parallel Sockets) which applications can use without the necessity of tuning the TCP window size and yet still achieve near optimal utilization of the network bandwidth. The idea is a simple one and an old one – we divide the data into partitions, open several sockets, and stripe the data over the sockets. We call this approach *network striping*. This can be broadly viewed as analogous to striping data over arrays of disks [9].

We believe that this paper makes the following contributions:

1. We describe five experiments demonstrating that network striping is an effective mechanism for applications to achieve high throughput on wide area high performance networks.
2. We describe a C++ library we have developed called PSockets that allows applications to quickly incorporate network striping with a few simple function calls.
3. We demonstrate that data intensive computing applications can easily incorporate PSockets into their design and transparently exploit wide area high performance networks.

We believe that this paper is novel for the following reasons:

1. With the exception of [8], of which we have just recently become aware, most approaches to improving performances have focused on system level tuning of TCP window size, not application-level network striping. In this paper, we show that this alternative approach is equally effective, and in certain situations, more effective, than tuning window size.
2. In [8], the data is divided into blocks and the blocks are striped across sockets for a satellite application. In our approach, given *n* sockets, we divide the data equally into *n* partitions and send the data asynchronously across the *n* sockets. This approach uses the TCP/IP stack to multiplex data across the different sockets.
3. Prior work on network tuning has focused largely on network measurements and not on application level measurements. In this paper, we also show that our

[1] 0-7803-9802-5/2000/$10.00 (c) 2000 IEEE.

approach using network striping allows the development of effective data intensive computing applications, an area of rapidly increasing importance.

4. We provide experimental data that shows in certain situations network striping using PSockets is more effective than tuning window size, even when theoretically this should not be expected. For example, it is common as the complexity of networks grows for there to be some hardware or software configuration errors somewhere along the path connecting the sender and receiver that limit network performance. This occurred quite often in our experiments. In this case, we have observed that network striping at the application level is still an effective mechanism for increasing network throughput, while; on the other hand, tuning window size is not nearly effective in practice.

This paper is organized as followed. PSockets is described in Section 3. Section 4 discusses four sets of experiments conducted over the Abilene network. Section 5 discusses some of the applications we have built using PSockets.

2. BACKGROUND

TCP was designed to operate reliably over almost any transmission medium regardless of transmission rate, delay, corruption, duplication, or reordering of segments [1]. TCP implementations currently adapt to transfer rates in the range of 100 bps to 10^7 bps and round-trip delay in the range of 1ms to 100 seconds [2]. TCP guarantees reliable in-order delivery of data sent from source to destination. Since many applications need reliable data delivery, TCP has been very widely used.

Typically an application requests data to be sent to another application at a remote machine. The TCP stack in the kernel of an operating system handles requests from various applications and passes data to the network. TCP partitions the data into *segments* to be sent over appropriate transmission media and waits for an acknowledgement from the receiver to know that a particular segment has been received correctly. TCP achieves this by having windows on both the sender and the receiver that can throttle the rate at which data can be sent [10].

Due to the recent advancements in fiber optics and related technologies, networks with very high transmission speeds are becoming more common. TCP was not designed for these types of networks [2] and recently there has been research addressing this issue [2, 3, 4, 5]. TCP window size is a key tuning factor for networks with large delays between the sender and the receiver. RFC1323 [2] provides details on achieving better TCP performance over such networks. Automatic TCP Buffer tuning to achieve increased bandwidth on a socket level is detailed in [3]. Improvements on the TCP algorithm to transmit data quickly and efficiently are explained in [4] and [5]. TCP performance improvements over satellite links were carried out in [8] which involved large delays. TCP performance depends not upon the transfer rate itself, but rather upon the product of the transfer rate and round-trip delay. This "bandwidth * delay product" measures the amount of data that would fill the pipe; it is the buffer space required at the sender and the receiver to obtain maximum throughput on the TCP connection over the established path, i.e., the amount of unacknowledged data that TCP must handle in order to keep the pipeline full. TCP performance problems arise when the bandwidth*delay product is large. This is referred to as "long, fat pipe" and a network containing this path is a long fat network (LFN). The three fundamental performance problems with the current TCP over LFN paths are the window size limit, recovery from losses and round-trip measurements. Several RFC have been developed and implemented to overcome these issues and achieve maximum performance with the current TCP over LFN.

The key factor to achieve maximum performance from TCP over LFN is the TCP window size. The TCP window size has to be tuned at both the source and the destination in order to achieve the maximum throughput. This involves changing parameters for the TCP stack in the kernel and has to be done by the system administrators at both ends [6]. Also, after the tuning, performance analysis has to be done to fine tune this window size. Typically this operation can take anywhere from few hours to even weeks [6]. In this paper we propose an alternative method involving striping data over several sockets. Applications can use the PSockets library and don't have to bother tuning the window size. Our library achieves equivalent performance to the one that would be achieved while setting the best window size.

A similar method with satellite applications in mind was proposed in [8]. Both use several open sockets. In [8], the data is divided into 8KB blocks and the blocks are striped across several sockets. In contrast, we have found it to be more advantageous to divide the data into equal partitions and send the partitions over the sockets asynchronously.

3. PSOCKETS

When an application needs to transfer data reliably over a network, it opens up a TCP/IP socket connection between the sender and the receiver. When the sender and receiver are connected by a high speed wide area network the bandwidth * delay product is quite high (greater than 64KB). In order to achieve the maximum TCP Performance on such networks the sender and the receiver's TCP window size is set to the bandwidth * delay product [2]. Enabling RFC 1323 on a system has to be done by the system administrators at both ends and not by the application. This could typically take a few days or sometimes even weeks. Hence the applications are limited to default maximum window size.

PSockets overcomes this limitation by using multiple sockets. Since the limitation of the TCP window size is only on a single socket, PSockets opens multiple socket connections

between the sender and the receiver. PSockets then divides the data to be transferred into equal partitions, where the number of open sockets determines the number of partitions. PSockets then sends the data partitions asynchronously over the multiple socket connections. The TCP/IP stack in the kernel multiplexes the data on the various connections, so that in effect the partitions appear to be sent in parallel on the multiple connections. Due to this effect PSockets is able to achieve the maximum data transfer rate possible on a high-speed wide area network without any network tuning. We call this approach *network striping*.

Figure 1 shows TCP packets between a source and destination pair at a given instant of time over a network. Assuming that the default TCP window size is set to 64KB and we have the bandwidth*delay product equal to 192KB we have shown three scenarios above for a LFN. In each case 192KB of data is to be transmitted from source to destination.

- Figure 1 (a): Here, the connection between the source and the destination has the default window size (64KB). At the source, the first 64KB of data is sent since the TCP windows size is 64KB. The sender then waits for an acknowledgement before sending any more data. Since the source waits for an acknowledgement from the destination the connection pipe between the source and destination is not completely filled. Therefore the maximum throughput attainable over the connection is not achieved. This limitation is overcome by using RFC 1323.
- Figure 1 (b): This diagram shows the connection between source and destination after enabling RFC 1323 which recommends the use of large TCP window sizes for connections having huge bandwidth * delay products. In our example the TCP window size is set to the 192KB (bandwidth * delay product). In this scenario, the source sends the entire 192KB of data to be transmitted to destination without waiting for an acknowledgement. Hence we can achieve the maximum data rate between the source and destination. The entire pipe is filled with segments up to the TCP window size.
- Figure 1 (c) shows our method in which we have opened multiple sockets. We do not have to do any network tuning for this scenario. Hence the TCP window size is set to the default value of 64KB. In this example, we have three sockets filling up the pipe since each socket has a window size of 64KB. With the help of three open sockets using the PSockets library we fill the LFN with the data to be sent and achieve equivalent performance of that shown in Figure 1 (b).

PSockets helps in application level network tuning to achieve the maximum throughput between a source and destination. The throughput attained by PSockets is equivalent to the one obtained over a fine tuned network which has a large bandwidth * delay product. If the TCP driver does not implement a good recovery algorithm during packet drops then the data in transition over the connection are flushed out and the retransmitted. This henceforth reduces the data rate on the connection. When the TCP window size is set to the bandwidth * delay product for the high-speed wide area network, the retransmission of transition data will be costly and affects the performance. By using PSockets, we can utilize smaller TCP window size and therefore during packet drops less amount of data retransmission. Therefore PSockets helps in achieving good performance even when there is deficiency in the TCP driver.

Using PSockets library, applications don't have to tune for the best TCP window size to achieve the maximum throughput. The library provided hides information on the number of sockets open, the segmentation, and the re-assembly of the data transferred. All the experiments detailed in the next section, include the overhead of segmentation and re-assembly. PSocket uses an API similar to the standard socket send and receive.

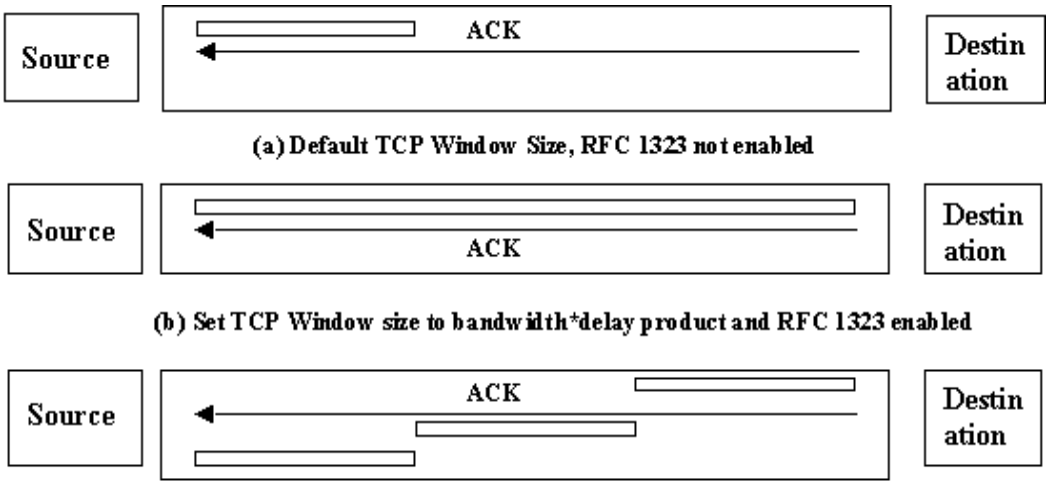

Figure 1: TCP packets at a given time over a LFN

4. EXPERIMENTAL RESULTS

Experiments were conducted between machines at Chicago, IL and Ann Arbor, MI. Table 1 provides the configuration of the machines. The connectivity between Chicago and Ann Arbor is an OC3 line (155 Mb/s). Even though we have an OC3 link between the two machines the limiting factor in this setup is the fast ethernet switch. Hence the maximum attainable throughput between the two machines is 100 Mb/s.

	Machine at Chicago, IL	**Machine at Ann Arbor, MI**
Processor	Dual processor Pentium II 450 MHz	Uni processor Pentium II 450 MHz
Network card	100 Mb/s fast Ethernet card	100 Mb/s fast Ethernet card
Operating System	Red Hat Linux 6.1	Red Hat Linux 6.1
External connectivity	ATM Switch	Fast Ethernet Switch

Table 1: Configuration of the Machines used for experiments

We performed several experiments to study the performance of PSockets. For all the experiments performed, the overhead in segmentation and reassembly of the data to be sent across the network has been included in our calculation. They are detailed below:

a) Varying number of sockets: The first test conducted was to vary the number of parallel sockets used for data transfer. The goal of the experiment was to find the optimal number of sockets for a particular buffer size. Figure 2 shows the results of our experiment for the TCP default window size of 64KB. The amount of data transmitted in each experiment was 64KB. It was observed that maximum throughput was obtained for a PSocket size of 12. Other experiments conducted (with a constant TCP window size and constant amount of data transfer) revealed that the maximum throughput was attained using Psocket sizes between 6 and 16. Using PSockets we were able to achieve a maximum throughput of 76 Mb/s of application data (not including other lower level overheads).

b) Varying the amount of data transfer: The goal of this experiment was to find the optimum data transfer unit for the maximum throughput. In this experiment the TCP window size was set to the default maximum of 64KB. Figure 3 shows the results of our experiments for data transfer amounts varying from 1KB to 4MB using PSockets as well as Iperf. It can be observed that the throughput increases with an increase in the number of sockets for a particular amount of data transfer and then finally drops down. This was also inferred from Figure 1. For a particular PSocket size the increase of data transfer amount from 1KB to 4MB did not have considerable effect of the throughput obtained. From this graph we can conclude that for a fixed number of sockets, varying the amount of data transfer did not affect the throughput.

c) Performance of PSockets with and without network tuning: The second test we performed was to vary the TCP window size and see the performance of using RFC 1323 as well as PSockets. To compare the performance of PSockets against standard software, we utilized Iperf [6], a network performance tool developed by NLANR. Figure 4 shows the results for two TCP window sizes 64KB (default TCP window size on Linux operating system) and 256KB (A TCP window size closer to the bandwidth * delay product between the machines used for the experiment). The data transmitted in each experiment was a 64KB. This clearly validates our argument that applications using PSockets can obtain throughputs equivalent to the one obtained after careful network tuning (RFC 1323) of the high-speed wide area networks. It can be observed that we are able to achieve a throughput of 57.7 Mb/s without network tuning and 67.7 with network tuning. We also observed that the maximum throughput attained using IPERF for a TCP window size of 256KB was only 47.7 Mb/s. This could potentially be a problem in the implementation of the TCP/IP driver or the network itself [7].

d) Performance of PSockets with various TCP window sizes: Figure 5 shows the performance of PSockets (8 sockets) for TCP window sizes 1KB to 4MB. The bandwidth * delay product for the connection is calculated to be 245KB. The amount of data transferred in each experiment was 8KB. It can be noticed from the graph that the maximum throughput was obtained for TCP window size 512KB. Hence as mentioned in [7], setting the TCP window size equal to the bandwidth * delay product actually does not give the best performance. The PSockets performance is compared against Iperf for the same window size. It can be observed that the difference in throughput obtained using PSockets and Iperf narrowed as the TCP window size increased. Since PSockets uses 8 sockets to send data, it performs better than Iperf up to the TCP window size of 8KB. Beyond the 8KB TCP window size, one would expect both PSockets and Iperf to be able to achieve the same throughput. If there is a problem in the implementation of the TCP/IP driver or with the network then Iperf would not achieve the maximum throughput with a single connection [7]. When Iperf was simulated with two clients at the same instance (similar to running PSockets with a value 2), this defect was confirmed since Iperf with 2 client connections was able to achieve an aggregate data rate more than Iperf with a single client.

e) Varying the number of clients. The next experiment we performed was to increase the number of applications running PSockets. We used a PSocket size of 8 for this experiment and a TCP window size of 64KB and the amount of data transferred by each application was 2MB. We observed that the aggregate throughput obtained by applications increased from 1 application up to 7 applications and dropped down for

8 applications. This is due to fact that the driver has to handle 64 sockets. This can be observed from Figure 6. It was observed that the throughput obtained by each application was nearly same indicating that each application was given an equal share of the bandwidth.

We also ran two more experiments to compare the results of applications not using PSockets with applications using PSockets. An application using PSockets with 8 sockets was run along with an application using a single socket. It was observed that the maximum attainable throughput was equally divided among the 9 sockets. Hence we can conclude that the applications not using PSockets are going to get lower throughput when run along with applications using PSockets. We analyzed the packet drops in two applications, which transferred 512KB of data – one using PSockets with 8 sockets and another regular application using 1 socket. The packet drop percentage in the two applications was 0.03% and 0.0002% respectively. This indicates that there is more contention within packets but we are able to achieve a very high throughput while using PSockets without the cost of network tuning.

From all of the above-mentioned experiments, we conclude that PSockets helps in achieving a throughput closer to the maximum throughput attainable over the network by using application level tuning rather than the more time consuming network tuning. PSockets would be able to help in achieving better throughput even when there are inherent problems with the driver or the network. Applications using PSockets will be able to extract a greater bandwidth share over a high-speed network while competing against other normal applications.

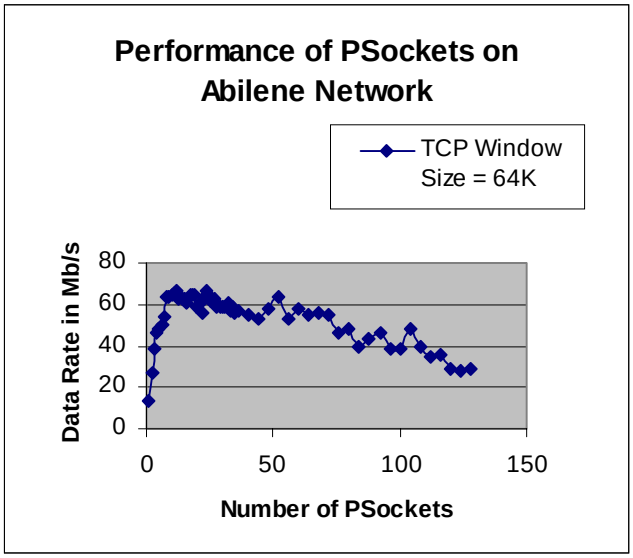

Figure 2: Performance of PSockets

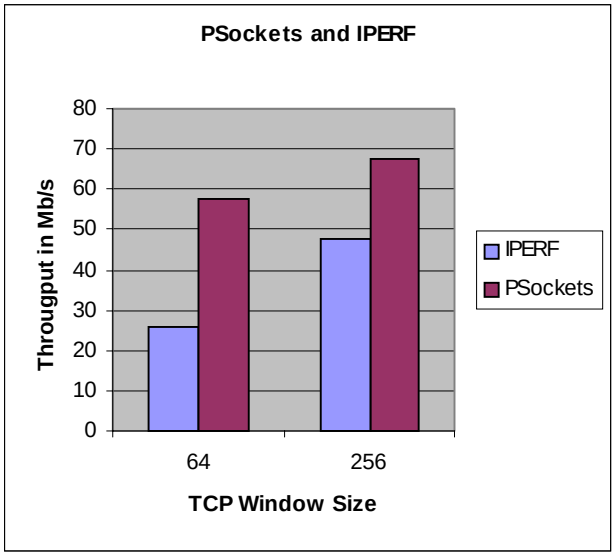

Figure 4: PSockets without and with network tuning

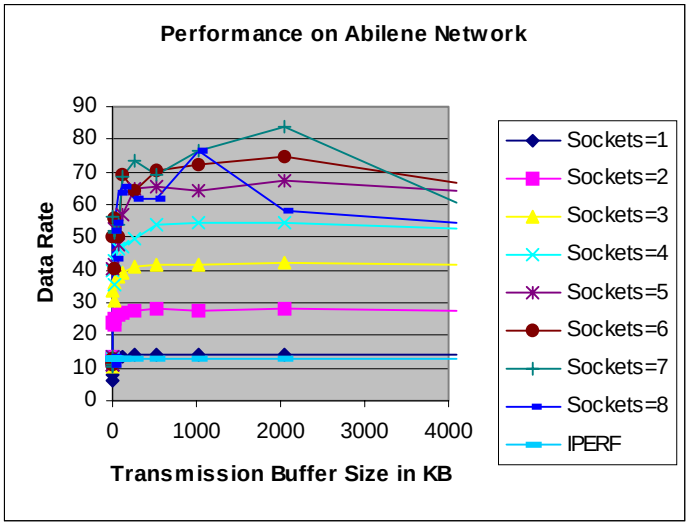

Figure 3: Performance of using Psockets on Abilene network for various transmission Buffer sizes

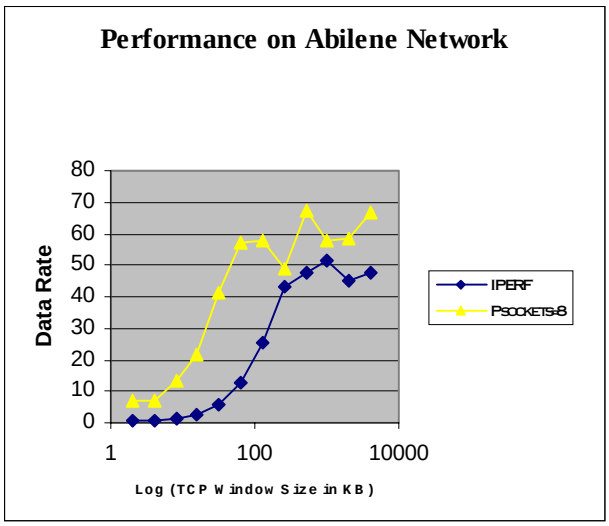

Figure 5: Performance Comparison of PSockets and IPERF with varying TCP Window Sizes

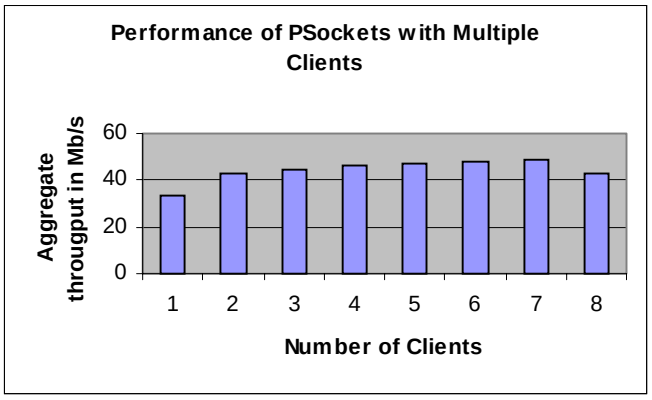

Figure 6: Performance of PSockets with Multiple Clients

5. APPLICATIONS

We built a geographically distributed data intensive computing application with PSockets to mine high energy physics data. The data was located at Chicago, Ann Arbor and Virginia. The Virginia link was limited due to a DS3 (45Mb/s), while the Ann Arbor was limited due to a fast ethernet switch of 100Mb/s. The Chicago machines were limited due to the OC3 link (155 Mb/). Six linux machines pulled data from these locations onto Portland, OR at the SuperComputing '99 conference floor. We were able to obtain a maximum throughput of around 30-35Mb/s while using the Physics application since the data was striped across 6 machines (2 machines at each site).

In addition, we developed a simple application benchmark program to test the raw performance of PSockets. We were able to achieve a maximum throughput of around 240Mb/s with the physics application and the benchmark program. The maximum theoretical attainable throughput was equal to around 300 Mb/s with all three sites. The input link at the floor was an OC12 (622 Mb/s). This demo was instrumental in achieving the "High Performance Communication Award" at the SC99 high performance computing challenge.

Currently we are using the PSockets library to build a high performance wide area data storage application called Osiris. Osiris stripes row-column data across various nodes distributed geographically and interconnected by a high speed wide area network. Osiris is aimed at applications that need large volumes of data to be retrieved quickly. PSockets is currently available for linux platform and can be downloaded from http://www.lac.uic.edu.

6. CONCLUSIONS

We have developed a library called PSockets, which helps wide area applications that need to move large amounts of data. Even though PSockets achieves the equivalent maximum bandwidth as that of having RFC 1323 enabled for TCP's performance, it is much easier to use and no tuning is required. Typically, tuning the network can be quite labor intensive, as such tuning requires system administrators on both ends as well as some performance study. With the PSockets library developers need not worry about this. PSockets hides the segmentation and re-assembly of data transfers over a wide area network using multiple sockets from the end user. Since PSockets has the same API as that of regular sockets it is very easy to use for application developers while at the same time achieving the maximum performance from the existing TCP stack without any performance specific tuning.

REFERENCES

[1] J. Postel, "Transmission Control Protocol", Request for Comments 0793, Sep. 1981.

[2] Van Jacobson, Robert Braden, and Dave Borman. "TCP extensions for high performance", Request for Comments 1323, May 1992.

[3] Jeffery Semke, Jamshid Mahdavi and Matthw Mathis, "Automatic TCP Buffer Tuning", ACM SIGCOMM, Oct. 1998.

[4] Experimental TCP selective acknowledgement implementations 1998, Obtain via: http://www.psc.edu/networking/tcp.html

[5] W. Richard Stevens, "TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms", Request for Comments 2001, March 1996.

[6] Mark Gates and Alex Warshavsky, Iperf version 1.1.1, Bandwidth Testing Tool, NLANR Applications, February 2000.

[7] Mark Gates, "Tuning Applications for High-Performance Networks", NLANR Distributed Computing Workshop, Sep 19-21, 1999, Tucson, AZ.

[8] S. Ostermann, M. Allman, H. Kruse, "*An Application-Level Solution to TCP's Satellite Inefficiencies*". Workshop on Satellite-based Information Services (WOSBIS), November, 1996. Rye, New York.

[9] D. A. Patterson, G. A. Gibson, R. H. Katz, "The Case for Redundant Arrays of Inexpensive Disks (RAID)", Proceedings ACM SIGMOD Conference, Chicago, IL, (May 1988).

[10] Wright Gray R., W. Richard Steven, "TCP/IP Illustrated, Volume 2 : The Implementation", Jan 1995, Addison Wesley.

ACKNOWLEDGMENTS

The authors thank Mr. Scott Wahlstrum and Mr. Marco Mazzucco for help in running certain tests for this paper.