Simple Available Bandwidth Utilization Library for High-Speed Wide Area Networks

M. Mazzucco, H. Sivakumar, Y. Pan, Q. Zhang
{siva, marco, jessie, qzhang}@lac.uic.edu
University of Illinois at Chicago
Chicago, IL, USA

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

Keywords: TCP, protocol, UDP, data mining, WAN, DSTP

December 21, 2001

Abstract

The growth of high-speed wide area networks (WANs) has enabled the emergence of a new class of data intensive, wide area computing applications, such as the remote analysis and exploration of data and data mining. The de facto standard for reliable data transfers is the Transmission Control Protocol (TCP). Despite improvements in TCP, over the years (to reduce overhead and achieve higher throughput) it is still a common experience for the end to end data rates of wide area data mining and data analysis applications to be disappointing. This paper introduces a library called SABUL (Simple Available Bandwidth Utilization Library) which merges features of UDP and TCP to produce a light-weight protocol with flow control, rate control and reliable transmission mechanisms designed for data intensive applications over wide area high performance networks. In this paper, we describe the library and give experimental evidence of its effectiveness.

1 Introduction

Although bandwidth is becoming a commodity, designing end to end network applications over wide area networks which can effectively use the available bandwidth is still a challenge. Our particular concern is with data intensive applications, such as remote data analysis and data mining [5], distributed data mining [4], and publishing experimental data over the web, such as required by large scientific collaborations, including EOS, the National Virtual Observatory, etc.

The overhead and inefficiencies of TCP/IP are well known, but it is the accepted transport protocol over the IP/routing solution since deploying TCP-based applications is easy. Furthermore, it is known that TCP's congestion avoidence mechanism works well, and thus it is strongly supported by the network community.

Since the original specification of TCP, RFC793, there have been several performance improvements suggested, such as RFC1122, RFC2001 and RFC2018. Despite these improvements, we argue in this paper

that there is still the need for a new protocol so that applications on high performance wide area networks can use available bandwidth transparently.

In this paper, we are concerned with developing a suitable protocol for fast, reliable large data transfers over wide area networks for data intensive applications. Data transfer methods to support distributed storage in local area networks are much better understood [6] and [9].

We have three requirements:

1. The protocol should not be greedy in its bandwidth use so that it can co-exist with all the other applications running on wide area networks.

2. Its performance should not be affected by the latency inherent in global wide area networks. One of our target applications is building a global data web and there will inevitably be latency moving data half way around the world.

3. It should be completely implemented and tuned at the application layer. To date, the successes in this area have generally required system level network tuning, such as kernel level adjustments of the TCP window size.

Applications such as video over IP and voice over IP are concerned with jitter or latency as they affect the quality of video or voice. However, these factors only concern us in as far as they affect throughput.

In order to satisfy these requirements, we decided to merge existing protocols to create a new protocol, SABUL, which can be simply used by any application, and which does not require any modifications to the kernel. Eventually, of course, the protocol could be implemented into the kernel, which may further improve its performance.

We believe that this work makes the following contributions:

1. Previous wide area data intensive applications have not effectively used the bandwidth of global high performance networks [4]. Data intensive applications built with SABUL can effectively use this bandwidth, as we demonstrate with our experimental studies.

2. Previous work has focused on using network striping to improve the end to end performance of wide area applications [3]. In this paper, we achieve similar performance by using UDP for a data channel and TCP for a control channel. More precisely, we use the TCP channel for rate control, flow control, and reliable transmission.

3. Our particular approach is optimized for wide area data intensive applications. Prior work of which we are aware has focused on local storage systems [9] and [6].

4. SABUL also continuously updates the communication state information using our TCP control channel. This allows us to more efficiently exploit the available bandwidth, even if it is dynamically changing.

We end this section describing some of our motivation for developing SABUL.

In prior work, we have developed an infrastructure called DataSpace [5] for creating a web of data. Although DataSpace was designed to work with high performance networks we observed that it was often difficult to use the bandwidth effectively. For this reason, we developed a library called PSockets [3] which used network striping at the application level to use the bandwidth effectively.

PSockets was first released in 1998 and since then has gone through several versions. We used PSockets during the Network Challenge at SC00 in November 2000. During the Network Challenge, we were connected to a router which was not configured properly and was performing poorly. On further investigation, with tools such as Iperf, we realized that the available bandwidth was reasonably high, around 400 Mb/s, but the performance from TCP/IP, even with the use of PSockets was not going over 225 Mb/s.

We investigated several possible causes for the poor performance. We first suspected the Gigabit Ethernet driver on Linux. Since we were able in the past to achieve better performance (with identical equipment and software) on a LAN, we eliminated this as a possible cause. Other possible causes were background packet losses and jitter, both of which we observed. By background packet losses we mean packet losses which are

independent of the rate of traffic flow. We assume that both of these were present due to the misconfigured router.

After SC00 we examined more carefully how the TCP stack was reacting to packet loss and latency issues while using the Psockets library. We summarized the results in the paper [3]. During these tests we also had an opportunity to test Psockets between two nodes, one located in Chicago, and the other in Amsterdam. This particular test further illustrated the problem that Psockets, and TCP in general, has with performing well over long distances, especially for data intensive applications. The results of this research helped us design SABUL. It was clear that rate control is necessary to achieve good throughput. Also, these measurements on parallel TCP sockets led us to the hypothesis that dynamically adjusting the rate in direct response to packet loss would lead to a protocol which was more "friendly" to TCP than TCP is to itself.

Section 2 contains background information. Section 3 introduces the protocol and Section 4 describes it in detail. Section 5 contains experimental results and Section 6 discusses them. Section 7 is the conclusion.

2 Background

The first computer network was developed by the Department of Defense. In 1969, ARPANET (consisting of four nodes) was in place and by 1973 there were connections to Europe. The transport layer protocol used for ARPANET was NCP (Network Connection Protocol). This was unsuitable to the needs of increasing traffic, so in 1974, TCP and IP were proposed. Since then, with minor modifications, TCP has been accepted as the standard protocol for reliable IP communication. Networks themselves, however, have changed drastically over the last twenty-five years. For example, they have become over 10^6 times faster and many times more reliable.

As performance requirements for TCP-based applications increased, several extensions to the protocol were proposed, and some of them have been widely accepted. One such extension allows for an increase in the window size (buffer size), along with a mechanism to control congestion. Effectively, this allows the rate of transmission to be independent of the RTT (round trip time) between two nodes. What is often referred to as "tuning the network" is setting the window size to RTT times available bandwidth. This should give a rate of flow which utilizes all the available bandwidth. In practice however, this is not the case. Mainly, this is due to the lack of rate control in TCP. Flow control, which is implemented by window size, ensures that the receiver of the IP flow is not flooded with more packets than it can process. However, the rate at which these packets are placed on the fiber is not dictated directly by TCP. This causes bursts in traffic flow, reducing the average rate of transmission. It also causes packet loss on the bursts.

This brings us to the now very well known formula:

$$Bandwidth = 1.3 * MTU/(RTT * sqrt(Loss))$$

[1] and [8], where MTU is the packet size used in the connection and Loss is the rate of packet loss in the connection. This has been shown to be a fairly good estimate of TCP traffic behavior. It also has become the criteria by which other protocols are measured in determining if they are "TCP friendly". It is important to note the role that RTT plays in the formula. The further away two communicating nodes are, the more they are penalized by packet loss in their communication. What makes this even more pertinent is that TCP is designed to increase the traffic flow up to the point where a packet loss is detected, at which point the flow is halved; this is then followed by a linear increase in flow until another packet loss is detected. This cyclic process is continued throughout the data transmission. Thus, by design, the throughput achievable by TCP between nodes separated by great distances is independent of the available bandwidth. Furthermore, no amount of "network tuning" can eliminate this simple fact.

Because of the lack of rate control in TCP, there have been various proposals over the years to develop protocols with rate control. In the multicast arena, there is SRM [2] which is gaining acceptance. With our current DataSpace application, we are only concerned with the unicast case. In this area there are several choices. Probably the most popular is XTP (Express Transfer Protocol). This is a rate based protocol which is a complete redesign of a transport layer protocol. It is not based on UDP or TCP, and includes a new header structure for IP packets. This needs to be implemented at the kernel layer, and thus is not currently of

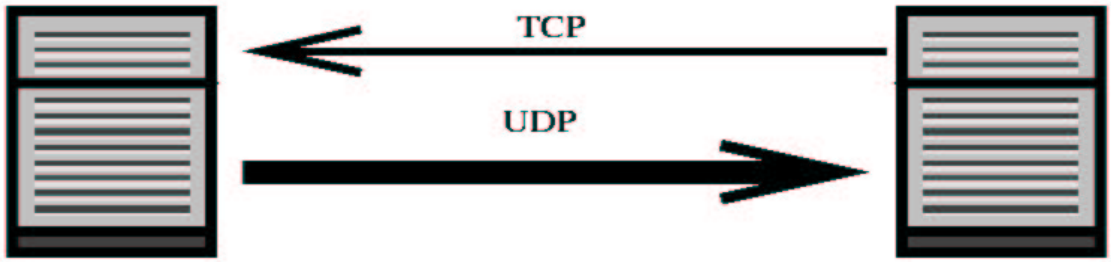

Figure 1: Overview of SABUL's architecture

interest to us. Of rate based protocols based on existing protocols, only NETBLT is RFC standardized. This was proposed in 1985, RFC969, but is not widely used. NETBLT can be described as a rate controlled UDP with error correction. It sends a block of data at a predefined rate. At the end of the initial transmission, NETBLT waits for a list of lost packets from the receiver in order to resends any lost packets. It was designed for unicast bulk data transfer, which is exactly what we need to drive our application. Our design of SABUL is similar to NETBLT, but with notable improvements: dynamic rate control and streaming data tranfers (not a block transfer protocol).

Another aspect of TCP which is often addressed in efforts to increase its performance is its overhead. There are three main sources for this overhead.

1. Acknowledgment packets: In order to implement the sliding window mechanism defined in the TCP specifications, an acknowledgment of every received packet needs to be transmitted to the sender. The size of these packets is relatively small, since there is no data payload.

2. Retransmission of a lost packet: In the original specification of TCP, as defined in RFC793, a lost packet results in the retransmission of all packets sent by the sender from the lost packet to the last acknowledged packet. This behavior has been eliminated in later implementations of TCP which implement RFC2018. In these implementations some form of selective acknowledgment is used, ensuring the retransmission of only the lost packet.

3. Size of the TCP header: This is of minor consequence compared to the other two causes of TCP overhead.

3 SABUL

In this section, we describe a network protocol called the Simple Available Bandwidth Utilization Library or SABUL.

The use of UDP in SABUL is a natural choice. Its header consists of destination and source addresses and has a field for checksum in most applications. UDP has no flow control, rate control, or reliable transmission mechanisms. Abstractly, it allows an application to work directly with the IP layer. It thus makes it straightforward to transform any protocol written on top of UDP into a protocol over IP in the kernel layer.

There is one fundamental novelty of SABUL compared with other rate based protocols over UDP, such as NETBLT, which is the use of both TCP and UDP in the protocol. The UDP channel is dedicated to the data transmission. The TCP channel is used for rate control, flow control, and reliable transmission. The packets on the UDP channel consist of the usual UDP header plus a 32 bit field for a sequence number. On the TCP channel, each packet consists of: a list of lost data packets, a field stating the requested data rate, and a field reserved to report the state of the receiver's available buffer size. We define the communication state information to be the information contained in these TCP packets.

The flow is assumed to be unidirectional. Data is sent to the receiver over the UDP channel, while current communication state information is sent over the TCP channel, from the receiver to the sender. The fundamental reason for this setup is simplicity. This simplicity allows us to design the protocol without concern for the potential loss of flow control, packet loss and rate information on the sender side. Since the communication state information is passed over TCP, its arrival is ensured; since the amount of this information is relatively small, it should have a negligible effect on the overall performance of SABUL. This simplification also allows us to easily make changes in the mechanism for rate control, packet loss recovery and flow control, since modifications are isolated to transmissions on the TCP channel.

Another reason for the use of both a TCP channel and a UDP channel is the need to write this protocol completely in the application layer. With two communication channels, the library used to implement the protocol on the sender side was designed to use two threads. One thread to listen for feedback from the receiver, and the other thread to transmit the data. This way, data can be sent at a controlled rate, not having to block for incoming packets containing communication state information from the receiver. Without the use of two communication channels, this could be avoided only if the protocol was implemented in the kernel. The library that we wrote to implement SABUL in the application layer uses the shared memory between the two threads to give the thread that is sending the data an updated account of the communication state information obtained from the thread listening to the receiver.

Thus, the main improvement of SABUL over NETBLT, and similar implementations of rate controlled reliable UDP protocols, is its continuous updating of communication state information. NETBLT uses a mechanism that sends buffers of data at a fixed rate. At the end of transmission of each buffer, the receiving side of NETBLT sends the sender a list of packets that were lost in the transmission of this buffer. The sender then resends these packets; the process continuing until all packets in the buffer are accounted for. Then the next buffer can be transmitted by NETBLT. NETBLT needs to block until all packets are accounted for on the sending side before sending another buffer. This process can be further delayed since packet loss information is transmitted unreliably by the the reciever to the sender (since this information is sent over UDP). Another deficit of NETBLT is that it needs to wait for at least one round trip time to get each update of packets lost.

In SABUL, however, each time the receiver notices at least one missing packet, it uses the TCP channel to transmit to the sender a list of packets that were lost. It does not have to block the sending of packets over the UDP channel to wait for an incoming packet containing the communication state information. This allows for changing the rate and flow of data, and retransmission of any missing packets during the transmission of the data. The list of missing packets is updated every time a missing packet is received. If during a predefined amount of time no packet was lost, and thus no transmission sent to the sender on the TCP channel, the receiver sends a notification of this fact to the sender with communication state information. This allows the sender to empty its buffer of packets which have successfully been received and adjust the rate and flow if necessary.

4 Using SABUL

An application using the SABUL library needs to use the Open, Send, and Close methods from the library, in this order, to send data over the network. An application needs to use the Open, Recv, and Close methods from SABUL to receive data. In the Open method, the sender and the receiver exchange information, over a TCP channel, about the maximum buffer size, initial data transfer rate requested, and the maximum transmission unit (MTU) size. The initial data transfer rate, as well as the MTU size on the sender, will be set to the minimum of the two rates advertised by the sender and the receiver.

In the Send method, a thread is initiated which will try to read the communication state of the receiver using the TCP channel. This thread is referred to as the TCP thread. The sender then starts by sending the data as UDP packets (equal in size to the MTU of the socket) in the main thread of the Send method (referred to as Main thread in this paper).

The Main thread has two phases. In the first phase, the Main thread sends the given data as UDP packets by attaching a sequence number using the "writev" call. The "writev" call is used since we do not have the overhead of attaching sequence numbers to the individual packets we send. After every packet, the Main

thread checks whether it is transmitting the packets at the initial rate. If it had transmitted the previous packet faster than the actual rate at which it's supposed to transmit, it waits for the calculated time interval before sending the current packet. After the transmission of a fixed number of packets (currently 500), it checks for the amount of packet loss obtained so far. After the same number of packets, the receiver sends the state information to the sender. This information is made available by the shared variable between the TCP thread and the Main thread. The Main thread then corrects itself to a data rate based on the feedback from the receiver. If there was no feedback from the receiver, the sender continues with the current data rate. The correction to a new data rate (higher/lower) is based on the amount of packet loss. The Main thread tries to tune itself to a packet loss rate of less than 1 percent. In the Main thread, after all data packets have been sent, then the Send method goes into the second phase.

In the second phase the Main thread fetches the sequence numbers of the packets that were lost in the first phase and starts sending those packets at the current data rate after the first phase. The Main thread also starts sending packets from a second buffer if it exists. That is if there was a subsequent send call, every alternate packet sent to the receiver is a packet from a second buffer. Note this is not a block transmission protocol for the following two reasons. First, the sender does not have to wait for the lost packet information, since it retrieved it over the TCP channel during the first phase. Secondly, a second buffer of data can be sent while the lost packets of the first buffer are retransmitted. As in the first phase, it corrects itself to a data rate for which the packet loss rate is maintained below 1 percent. The Main thread continues this process until all the packets have been received by the receiver. The receiver sends the information to the sender that all packets have been received, through the TCP channel.

The TCP thread gets the state information from the receiver. It does this by a blocking TCP read. There are four types of state information received by the TCP Thread:

1. Number of lost packets
2. The sequence numbers for lost packets
3. The total number of packets received
4. END signal indicating the completion of all data transmission

On the receiver side, the receiver keeps receiving the UDP packets and places them in the correct memory location determined by the sequence number and the MTU size. If a packet has been received correctly then the receiver attempts to receive the next packet. If the sequence number received is greater than the expected sequence number, then the receiver sends the list of lost packets (the packets between the expected sequence number and the obtained sequence number) to the sender through the TCP channel. If the sequence number is less than the expected sequence number, then the receiver places the packet in the correct memory location and removes it from the lost list. Once the receiver has received all packets, it sends an END signal to the sender.

Thus SABUL achieves the transmission of a large data transfer using a TCP and a UDP channel. The library uses the benefits of both UDP and TCP. The first implementation of this protocol achieves basic functionality. We are currently in the process of further developing the protocol to increase its performance. In addition, the next version of SABUL will include changes to the API allowing more flexibility in tuning at the application layer.

5 Experimental Results

The SABUL library was implemented in C++ on Linux operating systems. It has now been ported to various Unix platforms (SUN OS, Solaris, AIX, IRIX and Free BSD). The experimental results of SABUL on Linux and Irix machines are given below.

Tests were conducted between a node at Ann Arbor, MI and a node at NCAR, Boulder, CO. Both nodes used the Linux 2.2.17 kernel. They had 256 MB of RAM and 100 Mb/s fast ethernet cards. These two machines are interconnected by Internet 2's Abilene network with an OC-3 uplink (155 Mb/s). The bottleneck in this experiment is the Fast Ethernet cards.

We transferred 128 MB of data from Boulder to Ann Arbor. Results of SABUL are compared against Iperf as well as PSockets [3]. Iperf is a network performance monitoring tool from NLANR. Here we tuned the network with the bandwidth delay product which was calculated to be 512.5 KB. The PSockets library used finds the optimal number of parallel sockets and then transmits 128 MB. The packet loss was also measured during these tests using the "netstat" utility. The tests listed below are averaged over 5 runs.

Library/Utility used	Rate in Mb/s	Packet Loss Percentage
Iperf with TCP window tuning	83.3	0.0
PSockets with 19 parallel sockets	85.27	0.085
SABUL	95.63	0.95

Table 1

Both Iperf and PSockets use TCP for data transmission. Using Iperf along with TCP window tuning we obtained a throughput of 83.3 Mb/s. PSockets detected the best number of parallel sockets to be 19. The maximum throughput obtained using PSockets was found to be 85.27 Mb/s. The maximum packet loss percentage we have observed during a TCP transmission over a well tuned Abilene network was 1 percent. Therefore SABUL was tuned to run with a maximum packet loss of 1 percent. The percentage of packet loss is given in the last column. Table 1 shows that SABUL's performance is superior to the throughput results obtained by Iperf and PSockets.

Results of SABUL in the reverse direction, from Ann Arbor, MI to Boulder, CO are given in Table 2.

Library/Utility used	Rate in Mb/s	Packet Loss Percentage
Iperf with TCP window tuning	10.1	0.11
PSockets with 19 parallel sockets	40.68	0.57
SABUL	62.50	.4

Table 2

The throughput with the regular TCP channel with TCP window size tuning only was 10.1 Mb/s with a packet loss of 0.11 percent. The TCP window size was calculated to be 510 KB. PSockets with 19 parallel sockets produced a throughput of 40.68 Mb/s but the packet loss rate was found to be 0.57 percent. With the SABUL library we were actually able to achieve an average throughput of 62.50 Mb/s with maximum being 73.67 Mb/s and minimum being 31.36 Mb/s. Note that there were tremendous fluctuations in the packet loss while transmitting between Boulder to Ann Arbor. When necessary, the SABUL library auto-tuned itself to transmit at a low rate. The average packet loss while using the SABUL library was found to be .6 percent. The minimum and maximum packet loss percentages observed during these tests were 0.1 and 1.2, respectively.

Now we turn our attention to results which illustrate the congestion avoidance algorithm in SABUL. The graph in figure 2 shows the adjustments that take place in the transmission rate during a data transmission using SABUL. The measurements for both figures 2 and 3 were taken for one of the tests conducted from NCAR to Ann Arbor.

The graph in figure 3 illustrates how SABUL constantly adjusts its rate of transmission. It almost always (except during the initial rate adjustment) keeps the percent packet loss rate below the acceptable 1 percent throughout the entire transmission of the data buffer.

6 Discussion of Results

In general, we are encouraged by these results and believe that with further work SABUL can be an effective companion to PSockets. We realize, though, that more work on the algorithms in the library is required. The two algorithms in SABUL which need to be modified are the means by which packet rate is set, and the response of SABUL to packet loss.

During another set of tests, we requested a rate of 98 Mb/s in the LAN environment and achieved a rate of 98Mb/s with SABUL while PSockets reported a rate of 95 Mb/s. We predicted that we would achieve 98 Mb/s, or at least achieve a rate better than that obtained by PSockets because TCP has a greater overhead than the SABUL algorithm. That is, the packet loss in the LAN environment is negligible, and thus the

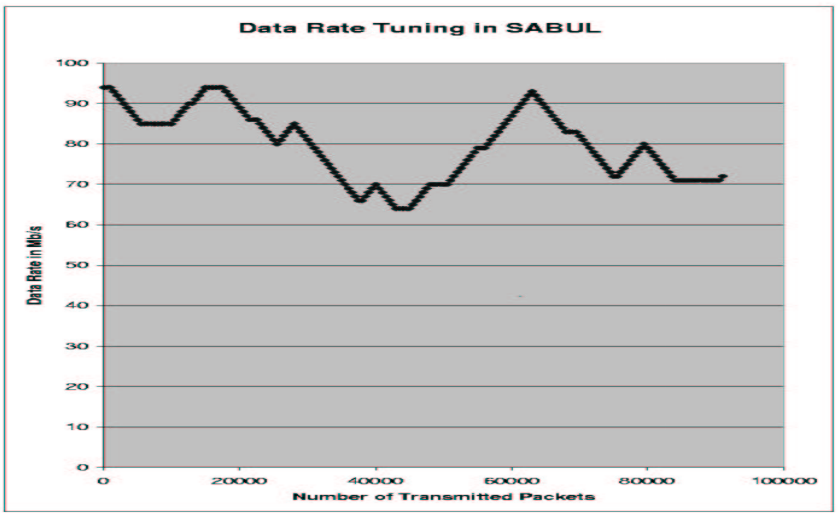

Figure 2: Varying data rate of SABUL

algorithm we use to deal with lost packets should have no effect. Thus in controlled conditions, the rate control algorithm seems to work.

However, we also observed on one occasion that SABUL gave unexpectedly poor performance in one direction (worse than TCP), between two nodes, while it gave good performance in the reverse direction. On further investigation it was observed that one of the nodes was already at 99 percent utilization. We hypothesize that the current rate control algorithm requires that the load on the node be somewhat less than full. We hope that with a better choice of algorithm, this can be avoided.

The tests we ran in the WAN environment showed that SABUL, in its current state, is able to obtain better performance than that of PSockets. We gauge performance as the ratio between throughput and loss data rate. The loss rate, in almost every test, was less than 1 percent. The reason for this is SABUL's dynamically adjusting rate control. However, there are times where it still goes over 1 percent, and we don't feel this is acceptable. In figure 3 we also see an initial spike in packet loss, but this we can attribute to setting the initial packet rate too high for the traffic conditions which where present during the test. We have never observed a single TCP connection to go over 1 percent, and thus hold this as a goal which needs to be met by SABUL. Note though, the PSockets library in general will not set itself to over 20 sockets, since this will not give the best throughput, but if it is forced to use 20, then there is an average packet loss of 5 percent [7]. That is, if there are 20 TCP connections into one machine competing for bandwidth, there is, on average, 5 percent packet loss. Since multiple SABUL connections could be tuned to keep the aggregate loss rate lower than one percent, it might be possible to show in the future that SABUL is even more "TCP friendly" than TCP.

To improve SABUL's response to packet loss, two aspects of SABUL's current implementation can be addressed. The first aspect is: as the packet loss goes over 1 percent, the rate of SABUL's transmission is reduced linearly. We will experiment with various other functions to reduce the transmission rate. The second aspect is: if the sender does not receive a rate notification from the receiver, it continues to transmit at the previous rate. However, it is possible that due to congestion there is a delay in the TCP channel which carries this information. We will experiment with reducing the rate according to various functions, if this information is not received from the receiver.

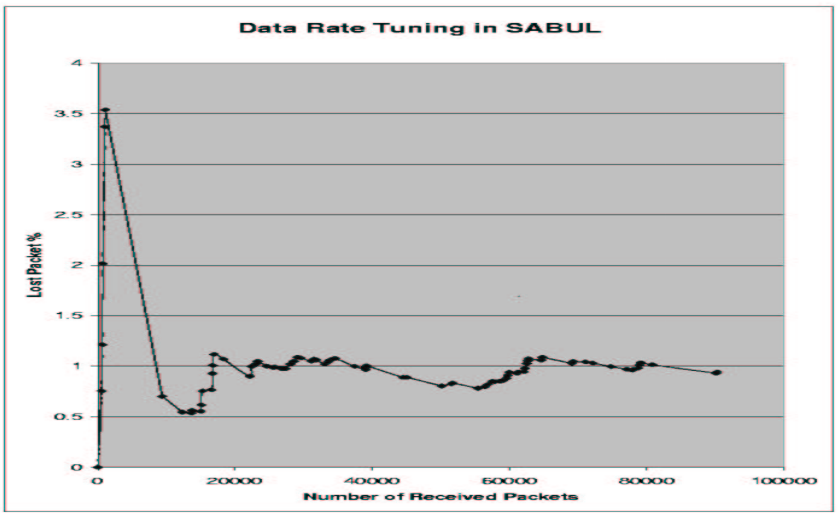

Figure 3: Varying packet loss percentage of SABUL

7 Conclusion

We are not proposing here that the days of TCP are numbered. Due to its wide acceptance and many refinements over the years, it should still serve well as the general purpose reliable transport protocol. We see SABUL as a more specialized protocol serving the needs of applications that require massive data transfer on high bandwidth networks over long distances.

The capabilities of both computers and networks have changed over the years since the conception of TCP. Modern networks have such large bandwidths that it has become a challenge to utilize the available bandwidth. Memory and CPU concerns, which affected the design of TCP, are different in the present environment. Modern day machines can store hundreds of megabytes in main memory and a program like SABUL takes advantage of this, allowing the application to set the size of the buffer. Presently this is being done in user space. However, if SABUL was used as a model for a future protocol written in the kernel then this could be done in Kernel space, saving time by reducing memory copying. Processor speeds have increased to the point where writing a rate control method is possible. Most modern day processors can do over 2 million instructions per second, so sending packets on the wire at a rate of 9 thousand packets (100 Mb/s) per second is not an issue. Even with the advent of Gigabit ethernet, if one uses jumbo packets, the wire rate still works out to only approximately ten thousand packets per second.

We are in the process of completing extensive testing on SABUL and the algorithm is being fine tuned, but the results obtained thus far are promising. If we take the performance of the protocol to be the ratio of transfer rate to packet loss, it has proven to be comparable to PSockets. The next step is to test it on Gigabit Ethernet and long distance transfers where we expect it to excel over PSockets. In [3] we have shown that PSockets performs at least as well as a well tuned TCP socket; thus, SABUL is a library which, in its initial conception, is meeting the performance of current TCP based protocols for the transfer of large data. In addition, like PSockets, tuning SABUL is done in the protocol, thus eliminating the extra time needed for tuning that occurs in applications using TCP directly. Finally, like PSockets, SABUL does not require system administrative privileges to facilitate tuning.

The design of SABUL, specifically the use of both TCP and UDP channels to keep protocol specific communications separate from data transfers, will allow us in the future to experiment with multiple algorithms. We are therefore very hopeful that we will be able to build a library which will exceed the abilities of any current library built solely on TCP or UDP for large data transfers.

References

[1] S. Floyd. Connections with multiple congested gateways in packet-switched networks part1: One-way traffic. *Computer Cummunications Review*, 21(5):30–47, October 1991.

[2] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, 1998.

[3] R. L. Grossman H. Sivakumar, S. Bailey. Psockets: The case for application-level network striping for data intensive applications using high speed wide area networks. *Proceedings of Supercomputing 2000*, November 2000.

[4] H. Kargupta and editors P. Chan. *Advances in Distributed and Parallel Knowledge Discovery*. AAAI Press/The MIT Press, Menlo Park, California, 2000.

[5] R. L. Grossman S. Bailey A. Ramu B. Malhi and A. Turinsky. In H. Kargupta and P. Chan, editors, *The Preliminary Design of Papyrus: A System for High Performance Distributed Data Mining over Clusters, in Advances in Distributed and Parallel Knowledge Discovery*, pages 259–275, Menlo Park, california, 2000. AAAI Press/The MIT Press.

[6] R. W. Watson R. A. Coyne, H. Hulen. November 1993.

[7] H. Sivakumar R. L. Grossman, M. Mazzucco. Behavior of transmission control protocol while using psockets over high speed wide area networks. *submitted for publication*.

[8] M. Mathis T. Ott, J. Kemperman. Window size behavior in tcp/ip with constant loss probability. *DIMACS Workshop on Performance of Realtime Applications on the Internet*, November 6, 1996.

[9] R.W. Watson and R.A. Coyne. The parallel i/0 architecture of the high performance storage system (hpss). *Proceedings of of the 14th IEEE Symp. Mass Storage Systems*, 1995.