SABUL: A Transport Protocol for Grid Computing

Yunhong Gu and Robert Grossman
Laboratory for Advanced Computing, University of Illinois at Chicago
*700 SEO M/C **249** 851 S Morgan St, Chicago, IL 60607, USA*
*E-mail: **yunhong@lac.uic.edu**; **grossman@uic.edu***

Key words: transport protocol, rate control, bandwidth-delay product, high performance data transport

Abstract

This paper describes SABUL, an application-level data transfer protocol for data-intensive applications over high bandwidth-delay product networks. SABUL is designed for reliability, high performance, fairness and stability. It uses UDP to transfer data and TCP to return control messages. A rate-based congestion control that tunes the inter-packet transmission time helps achieve both efficiency and fairness. In order to remove the fairness bias between flows with different network delays, SABUL adjusts its sending rate at uniform intervals, instead of at intervals determined by round trip time. This protocol has demonstrated its efficiency and fairness in both experimental and practical applications. SABUL has been implemented as an open source C++ library, which has been successfully used in several grid computing applications.

Abbreviations: SABUL – Simple Available Bandwidth Utilization Library, BDP – Bandwidth-Delay Product, RTT – Round Trip Time, AIMD – Additive Increase Multiplicative Decrease, MIMD – Multiplicative Increase Multiplicative Decrease, SYN – Synchronization, ACK – Acknowledgement, NAK – Negative Acknowledgement

1. Introduction

Although 1 Gbps and higher wide area networks are now becoming more common, it is still a challenge for grid applications to be able to use the bandwidth available, due to the limitations of current network transport protocols [26]. The limitations of today's network transport protocols are one of the main reasons that it is difficult to scale data intensive applications from local clusters and metropolitan area networks to wide area networks [2], [3], [7] and [26].

There are several problems that are the result of TCP's window-based congestion control algorithm when it is used on networks with high bandwidth delay products [26]. The bandwidth delay product is defined to be the product of the bandwidth and the round trip time (RTT) of a TCP packet. First, TCP's congestion control algorithm is not fair to flows with different RTTs [8]. Flows with high BDPs will generally get less of the available bandwidth than flows with lower BDPs when passing through a common bottleneck. In particular, the theoretical upper limit of throughput decreases as the BDP of the link increases [9]. Second, the AIMD

(Additional Increase Multiplicative Decrease) algorithm used by the congestion control algorithm to set the sending rate can take a very long time to discover the available bandwidth on high BDP networks [4, 5]. Third, dropped packets due to physical errors on the links but not due to congestion can prevent TCP from obtaining a high throughput [8].

A network transport protocol is considered *fair* if multiple flows can evenly share the available bandwidth. Since data intensive applications often use parallel flows to transport large data sets [7] or integrate data from multiple sources [27], fairness is important requirement for data intensive grid applications. The performance of these types of applications is often limited by the slowest data stream.

We performed a simple series of tests to determine the performance of TCP over a 1 Gbps link between Chicago and Amsterdam. The throughput of a single TCP stream without tuning the buffer size is about 4.5 Mbps. The throughput of a single TCP stream is about 30 Mbps when the TCP buffer is tuned by setting its size to 12 MB, the approximate BDP value of the link. With parallel TCP streams, a maximum throughout of 320 Mbps was obtained using 64 TCP flows, each with a 2 MB buffer. In another series of experiments, we set up two separate flows terminating in a common node in Chicago – one from a local node in Chicago and one from a remote node in Amsterdam. The former obtained 890 Mbps and the latter 3.5 Mbps.

These results show that TCP tuning or parallel TCP are not sufficient for data intensive grid applications. Several new transport protocols have been introduced to address these issues [12], [13], [14], [21], [22], [25], but few of them are widely used in grid computing. There are several reasons for this. First, many of the new protocols require that the existing network infrastructure be changed, for example, by requiring a specially-tuned operating system kernel or by requiring modification to the routers. There is a large cost for these types of changes, and problems may arise in collaborative networks. Second, some of the new protocols are fast, but not fair. This is adequate when a single flow is used for bulk data transport on a network without congestion, but not adequate for data intensive computing on grids, when multiple flows are needed. Third, some of the new protocols are not friendly to TCP. Although this is not a problem on specialized networks being used for bulk data transport, it is a problem for data grids in general. This is because grid and web services rely on TCP and when high volume flows impact the TCP flows, this can create problems for a data grid application.

In our opinion, there is a need for a network transport protocol which has high throughput (i.e. is fast), is fair, is friendly, and has low deployment cost. This motivated us to design and develop a reliable, high-performance, application-level, data transfer protocol named SABUL, or Simple Available Bandwidth Utilization Library. SABUL is an application- level data transfer protocol used to support high performance data transport in wide area lossy networks. It uses a UDP-based data channel and a TCP-based control channel. SABUL employs a rate-based congestion control mechanism. Both simulations and experimental studies demonstrate that SABUL is fair to other SABUL flows and is friendly to concurrent TCP flows.

We believe that the SABUL is novel in that it is one of the few network transport protocols we are aware of that can be deployed at the application layer, can efficiently use all available bandwidth on links with high BDPs, is fair to concurrent SABUL flows, and is friendly to concurrent TCP flows. We believe that data grid applications are more easily built using network protocols for high performance data transport with this combination of factors.

The rest of this paper will describe the details of SABUL. Section 2 describes some related work. Section 3 contains a description

of the SABUL protocol. Sections 4 contains some simulation results. Section 5 contains some experimental results. Section 6 contains some sample grid applications employing SABUL. Section 7 contains the conclusion.

2. Related Work

Network researchers have been improving TCP for many years, resulting in a series of TCP variations that include high-speed TCP [12], scalable TCP [13], and FAST TCP [25]. In addition to improving TCP, researchers have also introduced new network transport protocols designed to overcome TCP's inefficiencies on networks with high BDPs. Many of these new protocols, such as SABUL, are rate-based protocols which send packets at computed intervals instead of using windows.

Another approach is to design what are called open loop protocols in which routers provide information to the sender regarding congestion. Since this approach requires changing routers, it is not expected to be deployed for some time.

One of the goals of researchers designing variants of TCP is to retain compatibility with standard TCP. For this reason, these types of protocols typically modify just the sending algorithm. This is possible since congestion control based upon tuning the sending rate is done at the sender. The TCP stack on the receiving side does not have to be changed, simplifying deployment.

For example, Scalable TCP and HighSpeed TCP use more aggressive congestion control algorithms to replace the standard AIMD algorithm. As the congestion window becomes larger, they both increase faster; HighSpeed TCP also decreases slower, whereas Scalable TCP has a constant decrease factor of 1/8.

When the window size is below a threshold window size, they use the same algorithm as standard TCP. Because of this design, they are friendly to standard TCP for traditional low BDP networks; while at the same time, they are more aggressive over high BDP networks (BDP > threshold window size).

FAST TCP is a variant of TCP Vegas [19], which uses packet delay as an indicator of congestion in addition to packet loss. FAST TCP compares the current estimated RTT with the base RTT (defined as the smallest RTT ever observed), and tunes the window size using this ratio.

Window-based approaches to congestion control have a well known problem in high BDP networks: often by the time the sender knows there is congestion along the link, a great number of packets may have been dropped due to congestion at the gateways. Rate-based protocols are regarded as a solution to this problem. Rate-based protocols can found in NETBLT [9], VMTP [10], and more recently, Tsunami [14], RBUDP [22], and FOBS [21].

NETBLT is a bulk data transfer protocol that transfers data block by block. It updates data sending rates after each block using the packet loss of the last data block transfer. VMTP is a message transaction protocol that also uses rate control.

Recently several application layer protocols based upon UDP and TCP have been introduced for data intensive applications on grids, including SABUL, TSUNAMI, RBUDP, and FOBS.

RBUDP (Reliable Blast UDP), FOBS (Fast Object-Based Data Transfer System), and Tsunami are simple rate-based protocols based on UDP. Control information is sent over a TCP connection. In these protocols, application data is sent and acknowledged block by block, which can be thought of as a variation of selective acknowledgement. Since these three protocols lack congestion control mechanisms, so they can be only used in QoS enabled networks or in private networks.

As mentioned above, open loop methods have also been proposed for high performance data transport over networks with high BDPs. XCP [1] generalizes the Explicit Congestion Notification (ECN) protocol by using precise

congestion signaling in which the routers explicitly tell the sender the state of congestion and how to react to it. In XCP, routers monitor the input traffic rates of each of their output queues and tell the flows using the link to increase or decrease their congestion windows. This information is provided by annotating the congestion headers of data packets which are returned to the sender in the acknowledgment packet sent by the receiver. XCP uses a MIMD algorithm for controlling efficiency and an AIMD algorithm for controlling fairness.

None of the protocols mentioned so far are widely used today by grid applications; rather, grid applications today generally use TCP parameter tuning and/or parallel TCP streams. In particular, parameters controlling the buffer size or maximum window size are often changed since the default values are usually too small for high BDP links. Unfortunately, parameter tuning typically provides only modest improvements in throughput, and does not solve the fundamental problem of TCP congestion control.

Parallel TCP implementations, such as GridFTP [7] and PSockets [6], increase throughput by using multiple parallel TCP connections. Unfortunately in practice, parallel TCP usually requires extensive tuning [11], which can be quite labor intensive. In addition, parallel TCP exhibits performance and fairness shortcomings in lossy, wide area networks [11]. The Web100 and Net100 projects [18] are developing methods for automatic tuning which would overcome one of these shortcomings.

3. The SABUL Protocol

3.1. Design Rationale

SABUL is designed to transport data reliably. Since SABUL uses UDP for the data channel which is not reliable, this means that the SABUL protocol itself must detect and retransmit dropped packets. Using TCP as a control channel reduces the complexity of reliability mechanism.

To use available bandwidth efficiently, SABUL must be able to estimate the bandwidth available and recover from congestion events as soon as possible. The AIMD algorithm of TCP, which increases 1 byte for every RTT, is too slow for high BDP links; efficiency is improved by using a larger increase parameter as the bandwidth increases.

To improve performance, SABUL does not acknowledge every packet, but instead acknowledges packets at constant time intervals. This is a type of selective acknowledgement.

SABUL is designed to be fair to other SABUL flows so that grid applications can employ parallelism. Specifically, SABUL is designed so that all SABUL flows ultimately reach the same rate, independent of their initial sending rates and of the network delays.

SABUL is designed to be friendly to TCP flows so that it can be safely deployed on public networks.

SABUL is designed as an application layer library so that it can be easily deployed today, without requiring changes to operating systems network stacks or to the network infrastructure.

3.2. General Architecture

Figure 1 shows the architecture of SABUL. SABUL uses two connections: the control connection over TCP and the data connection over UDP. Note that the *data connection* is a logical and not a physical connection, since UDP is connectionless.

In this paper, for simplicity, we describe data transfer in one direction only. Data is sent from one side (the sender) to the other side (the receiver) using UDP. Control information is sent in the opposite direction from the receiver to the sender using TCP.

Memory management is handled as follows: The sender records each user buffer to be sent, but does not replicate it. The buffer

can be of any size. The receiver has its own protocol buffer to temporally store the received data before it is copied into an application buffer in a "recv" call.

To guarantee data reliability, both the sender and the receiver maintain a data structure to record the sequence numbers of lost packets. This data structure is called a loss list, and the sequence numbers are sorted in ascending order.

The sender transmits data packets at fixed time intervals set by the rate control algorithm. The time interval is called the inter-packet time. The sender transmits data packets as long as the total number of unacknowledged packets does not exceed a fixed window size, which can be set by the application. The receiver is responsible for receiving and reordering the data packets, detecting any lost packets, and sending back acknowledgements. The sender then adjusts the sending rate and updates the sender's buffer based upon feedback from the receiver.

3.3. Packet Formats

There are three kinds of packets in SABUL. The application data is packed in DATA packets, each with a 32-bit sequence number. The other two packets types are ACK, positive acknowledgement, and NAK, negative acknowledgement. The ACK packet tells the sender that the receiver has received all the packets up to the end sequence number. The NAK packet carries the number of lost packets and their sequence numbers (loss list).

All packets are limited to a MTU (Maximum Transfer Unit) size so that they will not be segmented automatically by UDP. Applications should try to send data in multiples of the SABUL payload [1] size if possible. However smaller packets are allowed, although some efficiency will be lost.

[1] Payload size is UDP MTU minus UDT header information.

Note that SABUL uses packet based sequencing, i.e., the sequence number increases by 1 for each sent packet.

3.4. Data Sending and Receiving

The sender always first checks the loss list when it is time to send a packet. If there are lost packets, the first packet in the list is resent and then removed. Otherwise, the sender determines whether the number of unacknowledged packets exceeds the flow window size, and if not, it packs a new packet and sends it out. The sender then waits for the next sending time decided by the rate control.

The flow window serves to limit the rate of packet loss due to congestion, when TCP control reports may be delayed. The maximum window size can be determined by the application: a good window size is the product of bandwidth and (SYN + RTT). Smaller values can limit the throughput. This is similar to the maximum window size in TCP.

After each constant synchronization (SYN) interval, the sender triggers a rate control event that updates the inter-packet time. The rate control algorithm will be introduced in section 3.5.

The receiver receives and reorders data packets. The sequence numbers of lost packets are recorded in the loss list, and removed when the resent packets are received.

If the sequence number of the current packet is greater than the largest sequence number ever received plus 1, then all the packets whose sequence numbers are between these two numbers are regarded as lost. If the sequence number of the current packet is not greater than the largest sequence number ever received, it is regarded as a retransmitted packet. Each of these conditions may be caused by out-of-order packets, but this is an infrequent condition.

The receiver sends ACK periodically if there are any newly received packets. The ACK interval is the same as SYN time. The higher the throughput is, the lower the ratio of

bandwidth that ACK packets should occupy. NAK is sent immediately once loss is detected. The loss will be reported again if the retransmission has not been received after k*RTT, where k is initialized as 2 and is increased by 1 each time the loss is reported. The increase of k reduces the frequency of repeated NAK's so that the sender is not blocked by the continuous arrival of loss reports. Loss information carried in NAK is compressed to save space for continuous loss.

In the worst case, there is 1 ACK for every received DATA packet if the packet arrival interval is not less than the SYN time. There are M/2 NAKs when every other DATA packet is lost.

3.5. Rate Control

SABUL's rate control tunes the inter-packet transmission time. However, if the number of packets sent per unit of time is regarded as a virtual window, this *window* control becomes a modified AIMD algorithm: the increase parameter is nearest power of ten that is not greater than the current sending rate; the decrease factor is a constant value.

The constant SYN interval in SABUL is 0.01 seconds. This number has been determined to be an acceptable trade-off between efficiency and fairness, between both intra-protocol and TCP. Larger values can cause SABUL to be less responsive to network change, slower in loss recovery, but more stable and friendlier to TCP; smaller values are the opposite. However, to achieve intra-protocol fairness, all SABUL implementations must use the same SYN value (i.e., 0.01 seconds).

Every SYN interval, the sender calculates the exponential moving average of the loss rate. If the loss rate is less than a small threshold (e.g., 0.1%), the sending rate is increased by the following scheme:

1. Calculate the increase in the number of packets to be sent in the next SYN interval with the following increase algorithm:

a) Calculate the current sending speed (S), in bps:

$$S = MTU * 8 / ipt$$

where MTU is in the unit of bytes, and *ipt* is the current inter-packet time.

b) Round S to the next highest power of ten (S_r):

$$S_r = 10\text{^}(ceiling(log10(S)))$$

c) Calculate the number of packets to be increased in the next SYN time (*inc*):

$$inc = S_r * SYN * beta / (MTU * 8)$$

where *beta* is the constant $1.2*10^{-5}$.

This formula gives a proper increase parameter relative to the current sending rate. For example, if the MTU is 1500 bytes, the increase parameters are:

1Mbps < S <= 10Mbps, inc = 0.01;
10Mbps <S <=100Mbps, inc = 0.1;
100Mbps < S <= 1000Mbps, inc = 1;
and so on.

d) inc is, at minimum, 1/MTU. (Note that 1 packet/MTU is 1 byte.)

2. The inter-packet time (*I*) is then recalculated according to:

$$SYN / I_{new} = SYN / I_{old} + inc$$

where I_{new} and I_{old} are the inter-packet time after and before this rate control, respectively.

The loss rate threshold allows some physical link error[2] when packet loss is not caused by congestion. This threshold should be higher than the physical link bit error rate but less than the average loss rate caused by network congestion. Here we assume that the loss rate caused by congestion in a short period is much larger than that caused by physical link error, otherwise SABUL will not work well. In fact, most transport protocols regard all loss as caused by congestion, which confirms that this assumption is reasonable.

[2] Loss of bits caused by network hardware, unrelated to congestion

The rate decrease algorithm is as follows:

1. The inter-packet time is increased by 1/8 as soon as the sender receives a NAK packet and:
   1) If the lost sequence number is greater than the largest sent sequence number when the last decrease occurred.
   2) If it is the 2^{dec_count}th NAK since the last time condition 1) was satisfied, where *dec_count* is set to 4, once 1) is satisfied and increased by 1, each time 2) is satisfied. That is, the series of thresholds are 16, 32, 64, etc.
2. The packet sending is frozen (no data is sent out) for a RTT once condition 1) is satisfied.

The objective of the increase scheme is to maintain an acceptable bandwidth distribution between coexisting TCP and SABUL flows, while allowing fast bandwidth discovery independent of network delay. However, it may lead to unfairness between flows with differing initial rates. This problem is alleviated by the decrease formula, supposing all flows that share the same bottleneck link have the same loss rate in the long run. Flows with higher sending rates will decrease more often. In addition, setting *dec_count* as 4 after the first decrease favors lower rate flows.

During high congestion periods, the sending rate may be decreased continuously. Meanwhile, the increase becomes slower as the sending rate decreases. The flow window limits the number of unacknowledged packets, and the feedback mechanism limits the frequency of control reports, so congestion collapse is avoided.

The flow window is 1 packet initially. Its size is increased to the number of acknowledged packets after each received ACK until it reaches the maximum window size or an NAK packet is received. Once the NAK is received, the flow window is set to the maximum value, and is not changed. The initial sending rate can be determined by application.

4. Simulation Analysis

We have built SABUL on a NS-2 simulator [24] to examine its efficiency and fairness. All of these simulations use a DropTail queue with queue size set to max (BDP, 10).

The simulation shows that SABUL can utilize available bandwidth efficiently, independent of bandwidth and network delay. The increase in transmission speed grows as the current sending rate increases. It will only decrease when necessary. Figure 2 shows the bandwidth utilization of SABUL under different link capacities and RTTs.

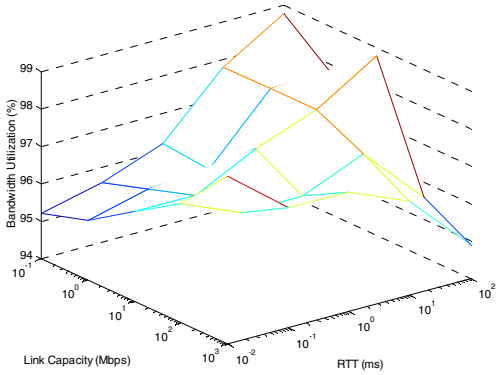

Figure 2. SABUL Bandwidth Utilization.

Intra-protocol fairness is examined by simulating SABUL flows with different initial sending rates and different RTTs.

The result is shown in Figure 3. This graph shows the performance of three simulated SABUL flows sharing a 1 Gbps link. The flows have different RTTs and different initial sending rates. The flows all converge at about 280 Mbps. This shows that the performance is independent of the RTTs and that it fairly distributes the available 1 Gbps bandwidth.

Unfairness may be caused by a flow having a higher than normal initial sending rate. However, this unfairness is limited by the decrease formula; flows with higher sending rates tend to suffer more loss, so the rate decreases will occurs more often.

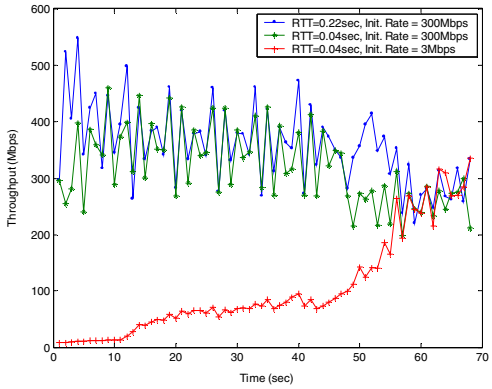

Figure 3. Intra-protocol Fairness.

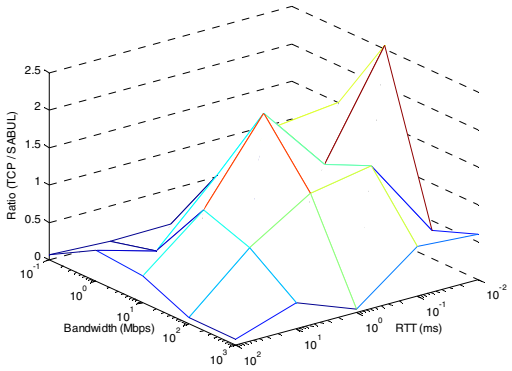

Figure 4. TCP Friendliness.

TCP compatibility is one of the objectives of SABUL. SABUL's major objective is to utilize bandwidth efficiently in high BDP environments where TCP is inefficient. In environments where TCP only uses a small percentage of the total available bandwidth, UDP will utilize the rest, while allowing TCP to transmit at its normal speed. On the other hand, on low BDP environments, SABUL and TCP should share the bandwidth fairly.

Figure 4 summarizes the results of simulations where TCP and SABUL flows share the same link for various different bandwidths and RTTs. In low BDP environments, TCP obtains more of the available bandwidth. In high BDP environments, where TCP is less efficient, SABUL is able to effectively use the available bandwidth.

This simulation uses TCP SACK with the maximum window size set to BDP, and the gateway uses a DropTail queue.

Besides efficiency and fairness, stability is also a consideration in SABUL. We discussed congestion collapse in section 3.5. Another concern is that delays in the TCP control connection might affect SABUL, since TCP has its own congestion control mechanism that can be affected by the data flow in UDP channels. In fact, this effect is so small that it can be ignored, because the number of packets in the control flow is far lower than the number of packets in the data flow.

The use of TCP for the control channel has a negative effect during high congestion. This is because the TCP channel, which only retransmits after a time out event, can delay the delivery of control packets. Due to the small number of control packets, the selective ACK mechanism of TCP [20] is not effective in this situation. This can cause SABUL to be less responsive to packet loss. But in a grid computing environment, where only a few data sources share the abundant bandwidth, such situation should seldom occur.

5. Experimental Results

In addition to simulation testing, we also tested SABUL experimentally. The testing was done using the NetherLight and CANARIE networks and the StarLight optical interchange facility. We used clusters located in Chicago, Ottawa, and Amsterdam for these tests.

All nodes in the clusters ran Linux Kernel 2.4 on dual Intel Xeon 1.8GHz CPU's. The TCP version used was TCP Reno with SACK option and ECN enabled. The TCP buffer was set to at least the BDP.

We first tested the end-to-end throughput of a single SABUL stream. We started 3 parallel SABUL flows from Chicago to Amsterdam. Each route had a 1 Gbps link capacity with 110ms RTT. The result is shown in Figure 5.

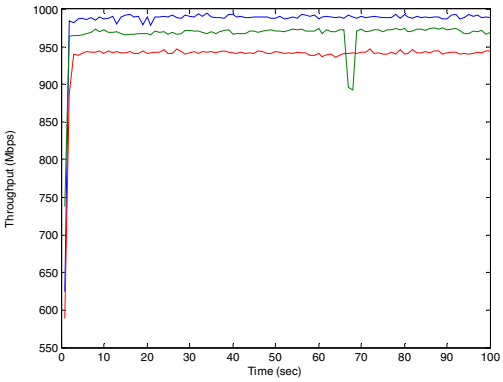

Figure 5. SABUL Trans-Atlantic Performance

At IGrid 2002 (3rd International Grid Conference), we successfully reached approximately 2.7 Gbps throughput on the same network (Table 1) [3].

Table 1: IGrid 2002Data Speed

SABUL 1	SABUL 2	SABUL 3	Total
902.8	902.9	907.1	2712.8

The relationship between SABUL and TCP, when sharing the same link in real networks, was also examined. In the first experiment, 2 TCP and 2 SABUL streams were started in a StarLight local network (Figure 6), where the link capacity was 1 Gbps and the RTT was 0.0004 seconds. TCP obtained slightly higher bandwidth than SABUL in this environment.

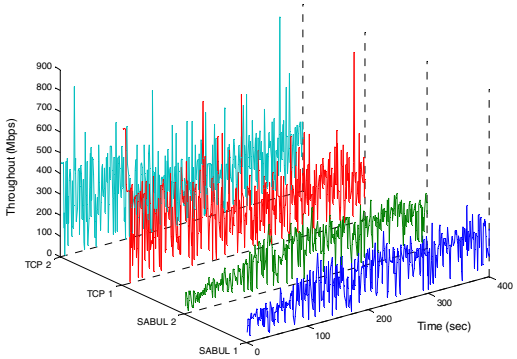

Figure 6. SABUL and TCP coexist on local high speed network

We also examined the impact of SABUL on a large amount of small TCP flows in which 1 MB of data was transferred in each connection. This experiment showed the performance of a very large number of small TCP flows on a 1 Gbps link connecting Chicago and Amsterdam in the presence of between 0 and 9 SABUL background flows. The result is shown in Figure 7.

The TCP version used in these experiments was SACK and the buffer size is set to at least the BDP.

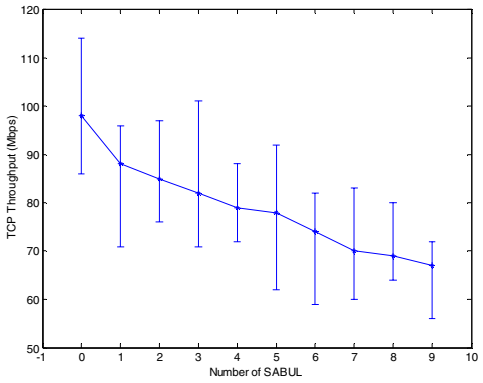

Figure 7. 500 TCP mice streams with SABUL background.

Disk IO makes high performance, end-to-end throughput more difficult since its IO speed is often bursty, and therefore not as continuous as a network interface; the disk IO speed is affected by many more factors, such as file system organization and disk scheduling. The bursty nature of the disk

requires the network protocol to quickly adapt to the end-to-end available bandwidth.

The results of end-to-end disk transport tests are listed in Table 2. Note that the major bottleneck is the disk IO and the synchronization between network interface and disk. With higher performance RAID disks, we expect that the end-to-end disk to disk performance would approach the end-to-end memory to memory performance on links with high BDPs, although the experimental testbed we used did not allow us to test this.

Table 2: File Transfer Performance

	StarLight (write 500)	Canarie (write 550)	SARA (write 800)
StarLight (read 800)	460	500	560
Canarie (read 800)	440	500	-
SARA (read 900)	440	-	600

6. Uses Case for Grid Computing

We have used SABUL for several grid-based data mining applications. In this section, we briefly describe three of these.

First, we have used SABUL in a grid application called DataSpace. DataSpace is an infrastructure for accessing, browsing, exploring, analyzing and mining remote and distributed data [17]. DataSpace uses standard TCP for moving small amounts of data and metadata, but it uses SABUL for moving large amounts of data and metadata.

Second, we developed an application called Lambda-FTP which uses SABUL for bulk data transfer [28]. Table 1 shows the results of using Lambda-FTP with three-way striping for transferring data across the Atlantic from Amsterdam to Chicago.

Third, we have also used SABUL to transport data for a distributed data mining application involving streaming data. In this application, two streams of data are merged using a best effort algorithm prior to the application of anomaly detection algorithm [15]. This application is called Lambda Merge.

Table 3. Lambda Merge Performance

window size (records)	Random (%)	Match (%)	Speed (Mbps)
10000	2	92	600
10000	10	82	630
10000	20	79	655
10000	33	71	671

Some performance data from the Lambda Merge application is shown in Table 3. This table shows the results of merging, in Chicago, 2 data streams originating in Amsterdam. The links had a maximum bandwidth of 1 Gbps. The goal of the application is to provide a best effort merge of two high volume data streams based upon a common key using a windowed, hash based algorithm; records can only be matched when there are both in the window. This is easy if the two data streams are ordered and synchronized, and becomes harder as the streams become less ordered and less synchronized. The table shows the results as the streams are randomized so that they become less ordered. When TCP replaced SABUL, the speed dropped to about 55 Mbps.

7. Conclusions and Future Work

SABUL is an application level library which is designed for data intensive grid applications over high performance networks. At the same time, it can coexist with TCP in both traditional, low BDP environments and high speed wide area networks. In both simulation and experimental studies, we have demonstrated that SABUL can efficiently use available bandwidth even on links with high BDP, that SABUL is fair to other SABUL flows, and that SABUL is friendly to concurrent TCP flows.

An initial version of SABUL was first developed in 2000 [16]. SABUL has now been implemented on a variety of UNIX

platforms and released as an open source project to the public [23]. SABUL has been used for several grid applications, including the DataSpace infrastructure for distributed data mining [17], Lambda-Merge for streaming data mining [15], Lambda-FTP for high performance bulk data transport [28].

We are currently working on several improvements to SABUL. There is a possibility of unfairness between concurrent SABUL flows, although it is limited to an acceptable ratio. In future work, we plan to improve the fairness of SABUL.

Another improvement we are working on is to use bandwidth estimation techniques to set the rate control parameters more efficiently. Finally, we are also working on designing a dynamic flow control mechanism instead of the fixed one that is currently used. For example, the window size used by the flow control can be dynamically set as the product of current available bandwidth and RTT.

Acknowledgements

This research work was supported in part by the National Science Foundation under grants number 0129609 and 9977868.

References

1. M. Hardley, D. Katabi, and C. Rohr., "Internet Congestion Control for Future High Bandwidth-Delay Product Environments." *Proceedings of ACM SIGCOMM 2002*.
2. I. Foster, C. Kesselman, and S. Tuecke. "The Anatomy of the Grid: Enabling Scalable Virtual Organizations." *International J. Supercomputer Applications,* 15(3), 2001.
3. R. L. Grossman, Y. Gu, D. Hanley, X. Hong, D. Lillethun, J. Levera, J. Mambretti, M. Mazzucco, and J. Weinberger. "Experimental Studies Using Photonic Data Services at IGrid 2002." *FGCS*, 2003.
4. S. K. Dao, E. Yan, and Y. Zhang. "A Measurement of TCP over Long-Delay Network." *Proc. of 6th Intl. Conf. on Telecommunication System.*
5. W. Feng and P. Tinnakornsrisuphap. "The Failure of TCP in High-Performance Computational Grids." Proc. of *Supercomputing 2002.*
6. S. Bailey, R. L. Grossman, and H. Sivakumar. "PSockets: The Case for Application-level Network Striping for Data Intensive Applications using High Speed Wide Area Networks." *Proc. of Supercomputing 2000.*
7. B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, and S. Tuecke. "Data Management and Transfer in High Performance Computational Grid Environments." in *Parallel Computing Journal,* Vol. 28 (5), May 2002.
8. T. V. Lakshman and U. Madhow. "The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss." *IEEE/ACM Trans. on Networking* 5, 3 (1997).
9. D. Clark, M. Lambert, and L. Zhang. "NETBLT: A High Throughput Transport Protocol." *Proc. of SIGCOMM '87, (Stowe, VT),* pp. 353--359.
10. D. Cheriton. "VMTP: Versatile Message Transaction Protocol Specification." RFC1045, April 1993.
11. T. Hacker, B. Athey, and B. Noble, "The End-to-End Performance Effects of Parallel TCP Sockets on a Lossy Wide-Area Network", in *Proc. of IPDPS 2002*.
12. S. Floyd, "HighSpeed TCP for Large Congestion Windows", draft draft-ietf-tsvwg-highspeed-01.txt, work in progress, 2003.
13. Scalable TCP, http://www-lce.eng.cam.ac.uk/~ctk21/scalable/, retrieved on 04/03/2003.

14. Tsunami, http://www.anml.iu.edu/anmlresearch.htm, retrieved on 04/07/2003.
15. M. Mazzucco, A. Ananthanarayan, R. L. Grossman, J. Levera, and G. B. Rao, "Merging Multiple Data Streams on Common Keys over High Performance Networks", in *Proceedings of SC 02*.
16. H. Sivakumar, R. L. Grossman, M. Mazzucco, Y. Pan, Q. Zhang, "Simple Available Bandwidth Utilization Library for High-Speed Wide Area Networks", in *Journal of Supercomputing*, 2004, to appear.
17. R. Grossman and M. Mazzucco, "DataSpace - A Web Infrastructure for the Exploratory Analysis and Mining of Data", in *IEEE Computing in Science and Engineering*, July/August, 2002, pp. 44-51.
18. T. Dunigan, M. Mathis and B. Tierney, "A TCP Tuning Daemon", in *Proc. of IEEE SuperComputing 2002*.
19. L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance", 1994 ACM SIGCOMM Conference, pages 24 - 35.
20. Mathis, M., Mahdavi, J., Floyd, S., and Romanow, A., "TCP Selective Acknowledgement Options", RFC 2018, April 1996.
21. P. Dickens, "FOBS: A Lightweight Communication Protocol for Grid Computing". *Europar 2003*.
22. E. He, Leigh, J., Yu, O., and DeFanti T. A., "Reliable Blast UDP: Predictable High Performance Bulk Data Transfer", in *IEEE Cluster Computing 2002*, Chicago, IL, Sep. 2002.
23. SABUL source code, http://sourceforge.net/projects/dataspace, retrieved on 10/03/2003.
24. NS-2. http://www.isi.edu/nsnam/ns/.
25. FAST TCP. http://netlab.caltech.edu/fast, retrieved on 10/03/2003.
26. A. Chien, T. Faber, A. Falk, J. Bannister, R. Grossman, J. Leigh, Transport Protocols for High Performance: Whither TCP?, Communications ACM, Volume 46, Issue 11, November, 2003, pages 42-49.
27. Marco Mazzucco, Asvin Ananthanarayan, Robert L. Grossman, Jorge Levera, and Gokulnath Bhagavantha Rao, Merging Multiple Data Streams on Common Keys over High Performance Networks, Proceedings of the IEEE/ACM SC2002 Conference, 2002, IEEE Computer Society, page 67.
28. Project DataSpace, Lambda-FTP, retrieved from http://sourceforge.net/projects/dataspace on December 10, 2003.

[1] Sent and acknowledged data; [2] Send but not acknowledged data; [3] Unsent data; [4] Empty buffer area that new data can be written; [5] Acknowledged data that can be read by application; [6] Dirty data area (unacknowledged).

Figure 1. UDT Architecture.