SABUL: A High Performance Data Transfer Protocol[1]

Yunhong Gu ygu@cs.uic.edu
Xinwei Hong xwhong@lac.uic.edu
Marco Mazzucco marco@dmg.org
Robert Grossman[2] grossman@uic.edu

Laboratory for Advanced Computing, University of Illinois at Chicago
M/C 249, 851 S Morgan St, Chicago, IL 60607

Abstract

The paper describes a general purpose high performance data transfer protocol for data intensive applications over high bandwidth networks. The protocol, named SABUL, has demonstrated the efficiency and fairness features in both experimental and practical applications. SABUL is a lightweight, reliable, application level protocol. It uses UDP to transfer data and TCP to feedback control messages. The protocol uses a rate based congestion control that tunes the inter-packet time. This algorithm is proven to be TCP friendly. In addition, the protocol also specifies the transparent memory copy avoidance.

Keyword: SABUL, high performance protocol, rate control, optical network, memory copy avoidance

1. Introduction

Photonic technology has pushed the network bandwidth to 10Gbps and enabled a wide range of new, powerful applications, including high resolution streaming media, remote access to scientific instruments, and specialized virtual-reality such as tele-immersion, etc. As the rapid increase of the network bandwidth and the emergence of new routing/switching technology like MPLS, the end systems and the data transfer protocols are becoming bottlenecks in more and more scenarios. This phenomenon occurs particularly often in scientific computing networks, grid networks [10], high speed WANs, etc. where a few sources share the abundant optical bandwidth.

IP protocols are still the predominant in these networks. However, the overhead and inefficiencies of TCP prevent its use in this kind of applications. The drawbacks of TCP inherent in its window based congestion control mechanism have been well known [7]. The situation is particularly serious in high bandwidth long delay networks because the TCP throughput is inversely proportional to the network delay. Furthermore, if the bandwidth is very high, the increase of window size will last a very long period, and once the loss occurs the window size reduces to half, which makes the situation more serious.

There are more problems for TCP. Connections with short delay will grab more bandwidth when competing with long delay connections due to the congestion control algorithm described above. If the network situations are not symmetry, the bandwidth utilization will also be limited by the slow direction. The acknowledgement for each received packet is also a high overhead over high bandwidth where tens of thousands of packets are sent out per second.

Network researchers have come up with some solutions. These solutions include improvement to TCP, new transport protocols such as XTP [12], and application level solutions. To improve the TCP protocol is not an ultimate solution because it cannot change the window based mechanism. And new protocols at the same level as TCP are hard to gain popularity because of the hardware and software cost. Although new switching, routing, and transportation protocols will definitely replace or enhance the current network protocols in future, this is not going to happen soon.

[1] This work is supported by grants from NSF under contract No. ANI-9977868.
[2] Robert Grossman is also with Magnify, Inc.

This background motivated us to design and develop a lightweight high performance application-level transfer protocol. This new protocol is called SABUL, or simple available bandwidth utilization library. It uses a rate-adjusting UDP connection to transfer data and a TCP connection for feedback.

The rest of the paper will examine the detail of the SABUL protocol. Section 2 introduces the design objectives of the protocol. Section 3 describes the SABUL protocol in detail. The experimental results will be discussed in section 4. We give some successful application examples that use SABUL as data transfer protocol in section 5. The related works are reviewed in section 6 before we conclude the paper in section 7 with a brief look at the future work.

2. Design Rationale

The overall objective of our new protocol is to provide a general purpose data transfer service that can utilize the bandwidth efficiently and fairly.

The requirements come from the situation of huge amounts of data transfer over high speed network. However, the new protocol is not dedicated to bulk data. It can transfer data from one single byte to hundreds of gigabits. The interface should be the same, or at least similar as the general socket API. By "general purpose" it also means that the protocol should not modify the current network infrastructure such that it can be deployed with lowest cost.

It is natural that to use UDP with loss correction and rate control for the new protocol. UDP provides a transportation level interface of IP with error detection. This flexibility is suitable to build new protocol at application level. Since reliability is also needed, we use TCP to transfer control information.

The efficiency objective requires that the protocol is simple and lightweight. The protocol header and computation overhead should not cost too many system resources. The overhead of control packets should also be as small as possible.

The efficiency objective also requires that the protocol should adapt itself to the network change automatically and rapidly (convergence). A rate control mechanism with rapid reaction and little oscillation is required. In addition, the algorithm should be stable to any network changes.

We also need a high performance memory management module for the protocol. Since there may be hundreds of thousands of packets being sent and received every second, the data move in the memory is a critical path in a practical system. Data replication should be reduced to minimum. However, any method that leads to changes of the API semantics is not desirable because of the cost of application development and immigration cost it causes.

The fairness objective needs a fair congestion control algorithm. First of all, fairness means different connections sharing the same network should have same throughput. Second, it should be friendly to other friendly protocols, e.g. the TCP.

We summary the seven objectives below:

- Provide reliable data transfer (reliability).
- The protocol can be implemented at application level.
- Packet and computation overhead should not be too large to prevent the performance (lightweight).
- Fully utilize the available bandwidth (efficiency).
- Adapt the network bandwidth automatically and rapidly and be reliable to network change (convergence).
- Share the bandwidth fairly with other connections (fairness).
- Manage the protocol/application memory efficiently and transparently (memory copy avoidance).

3. Principle Protocol Specifications

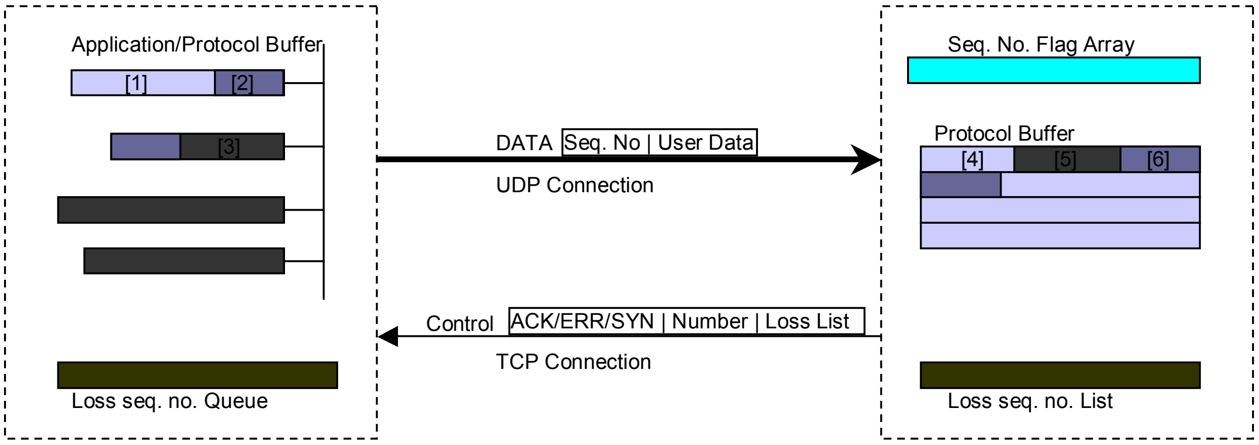
3.1 General Architecture

SABUL uses two connections: the control connection over TCP and the data connection over UDP.

A SABUL connection is uni-directional: data can be only sent from one side to the other side. According to the relationship between the two sides involved in SABUL, we call them the sender (side) and the receiver (side), respectively. The sender initializes the connection and waits for the receiver to connect to it and then constructs the control connection. The data connection is built up following a successful control connection.

Data flow is from the sender to the receiver only. The control information is from the receiver to the sender only, so we also call the control information as feedback.

The sender manages application buffer and is responsible for its transmission, retransmission, and release according to the feedback from the receiver. It manages a queue to record the lost packets. The receiver reorders the packets according to the sequence number and puts them into its own buffer or the application buffer. A flag array is kept for packet reordering and loss detection. The sequence numbers of the lost packets number are kept in a lost list.



[1] Acknowledged Data; [2] Sent but not acknowledged; [3] not sent; [4] Unwritten area; [5] Written and acknowledged area, data can be read; [6] Written but not acknowledged area.

Figure 1. SABUL Protocol Architecture

3.2 Packet Formats

There are four kinds of packets in SABUL. The user data is packed in DATA packet and sent through UDP connection from the sender to the receiver. The other three are feedback control information from the receiver to the sender through TCP connection. ACK packets are positive acknowledgement telling the sender that the receiver has received all the packets till the sequence number carried in the ACK packet. ERR packets are negative acknowledgement that carry the sequence numbers of the lost packet. And an SYN packet is a synchronization packet with the packet receiving rate and triggers a rate control events in the sender side.

Sequence Number	Data
32-bit	1 – 1468bit

Figure 2. SABUL data packet structure

Packet Type	Attribute	Loss List
32-bit	32-bit	4x $(0 \le x \le 357)$

Figure 3. SABUL control packet structure

3.3 Data Sending/Receiving Algorithm

The sender has two basic functionalities: to send out packet periodically according to the rate control mechanism and to process the feedback from the receiver. The retransmission of a lost packet has higher priority than the first-time sent packet, otherwise the unacknowledged packets will be kept in the buffers of both sides and prevent further packet transmission because the buffers have limited lengths. The processing of feedback should be synchronized with data sending. In addition, the sender is responsible for timer expiration detection.

Data Structure:

1. The protocol buffer is linked list whose nodes are the head pointers of application buffers.
2. The loss queue is a FIFO queue that stores the sequence numbers of the lost packets from ERR feedbacks.

Algorithms:

```
STEP 1.   Initialization:
STEP 2.   Get current time;
STEP 3.   Poll Control Channel. Receive control packet and process it if there is any. Otherwise, calculate the time passed since the last ACK or ERR feedback packet was received, and if the interval is greater than the time expiration threshold, generate an EXP event and process it;
STEP 4.   If the loss queue is empty, GOTO STEP 5. Otherwise, read (and remove) a sequence number and retransmit the proper packet, then GOTO STEP 6;
STEP 5.   If the protocol buffer is not empty and the difference between the largest sent sequence number and the last acknowledgement number is less than a valve, send a new packet and update the most recent sequence number;
STEP 6.   Block the sending for an inter-packet time interval and GOTO STEP 2.
```

Figure 2. Data Sending Algorithm

The receiver is responsible to reorder the received packets, detect the loss, and feedback error, acknowledgement and synchronization messages.

Data Structure:

1. The protocol buffer is a logically circular memory block for temporally storing and reordering the received packets.
2. The offset array is a flag array that records the packet with which offset (difference between the sequence number and the last acknowledgement number) is received or not.
3. The loss sequence number list is a linked list whose node is a structure of a lost sequence number and the last error report time of the sequence number.

Algorithm:

```
STEP 1.   Initialization:
STEP 2.
          a. Get current time;
          b. If the time passed since the last acknowledgment event exceeds the ACK interval, or the user registered buffer has been fulfilled, generate an ACK packet;
          c. If the time passed since the last error feedback event exceeds the ERR interval, and the loss list is not empty, generate an ERR packet;
          d. If the time passed since the last synchronization event exceeds the SYN interval, generate an SYN packet;
STEP 3.   Start a timed receive; if the timer is expired, GOTO STEP 2;
STEP 4.   Read the sequence number in the data packet, and calculate the offset since the last acknowledged sequence number;
          a. If the offset is greater than or equal to the sequence number array size, generate an ACK packet, and GOTO STEP 2;
          b. If the offset is less than 0, GOTO STEP 2;
          c. If the offset is greater than expected offset, insert sequence number from current largest received sequence number to the current received sequence number to the loss list, generate an ERR feedback.
STEP 5.   Update the next expected offset number, which is the smallest number since the most recent offset (current) in offset flag that hasn't been set;
STEP 6.   Update the largest received sequence number;
STEP 7.     Set the proper flag in the offset array, and GOTO STEP 2.
```

Figure 3. Data Receiving Algorithm

3.4 Control Information Generation and Processing

The generation and processing of control information is the kernel part that keeps the protocol running correctly. As we have mentioned above, control information is generated at the receiver side and processed by the sender. The data structures used in this section is the same as the previous section.

All the three kinds of feedback are generated based on their own timers. In addition, a detection of packets loss can generate immediate ERR feedback, and an ACK packet can be generated when the boundary of a user buffer is reached (see section 3.4).

ACK Generation: If the loss list is empty, the ACK sequence number is the largest received sequence number plus 1; otherwise, the first sequence number in the loss list is the ACK sequence number. The feedback is not sent if the sequence number to be acknowledged is the same as the previous.

ERR Generation: Every node in the loss list is attached with a timestamp. During the ERR generation, only the sequence number on which node the difference of current time and the timestamp is greater than a threshold will be added into the ERR packet. The feedback is not sent if no sequence number is added into the packet.

SYN Generation: The SYN feedback simply sends back a packet with SYN type information followed by the total number of received packets since the last SYN event.

Feedback processing is an independent part in the sender, however, it is synchronous, i.e., when feedback processing is called, the sender will block the data sending until the processing finishes. The sender itself will generate an event of EXP when it doesn't receive any feedback except for SYN from the receiver before a timer is expired.

ACK processing: ACK processing sends a message to the sender buffer telling the latter to modify the acknowledgment pointer and free user buffer if all data in that buffer are acknowledged. ACK processor calculates an estimated acknowledge size by simply multiplying numbers of packets by the fixed packet size. When the buffer processing the acknowledgment, it needs to adjust the size when it reaches the end of a buffer and the last packet is less than the fixed packet size.

ERR processing: ERR processor inserts the sequence numbers of lost packets to the loss queue. The numbers of ERR packet is updated for rate control use (section 3.6).

SYN processing: SYN processing triggers a rate control event (section 3.6).

EXP event processing: EXP means all the packets since the last acknowledgment packet were lost. EXP processor adds these sequence numbers to the loss queue.

3.5 Memory Management

The sender side buffer is a list of application data buffers. Each node of the list (a block of buffer) is the data block that the application calls SABUL to send. The buffer is released by SABUL after it is been sent successfully.

There are four pointers in the sender's buffer: the current ACK block pointer (CA), the current ACK pointer (CAP), the current sending block pointer (CS), and the current sending pointer (CSP). When CAP is greater than or equal to the end of CA, CA is freed and moved to next block if there is one, and CAP are calculated to represent the position in new CA. CAP is updated by the processing of ACK feedback. CA and CAP are also used to find a data position to be re-transmitted. CS and CSP are used to pack new packet (first time sent packet).

SABUL always tries to read data in fixed size and shape fixed-size data packets. However, if the last packet of a buffer is less than the desired size, SABUL sends out a smaller packet and the receiver will find it out through the packet size.

The receiver side buffer is a block of continuous memory, and is logically circular. Two pointers tell the head (the further position of new data, i.e., the data that has the largest sequence number currently) and the tail (from which point the application can read data), respectively. An additional pointer is used to record the position before which all the data was received correctly – the acknowledgment pointer.

One of the goals of the design of the receiver management is to provide transparent memory copy avoidance between protocol memory and application memory. Transparent data copy avoidance means avoiding data copy without change the API semantics. SABUL uses a method called *user buffer registration.*

When an application calls "recv" method to fetch data, SABUL inserts the user buffer to the protocol buffer logically. Before that, all the data from the acknowledgment point to the head point is copied into the user buffer. The insertion of the user buffer is equivalent to extending the protocol buffer by the user buffer size. At the same time, the acknowledgment point and head point of the original buffer are set to the tail buffer, if they are less than the size of user buffer; otherwise, decrease them from the user buffer size. When the user buffer is fully filled, application call will be returned with the buffer.

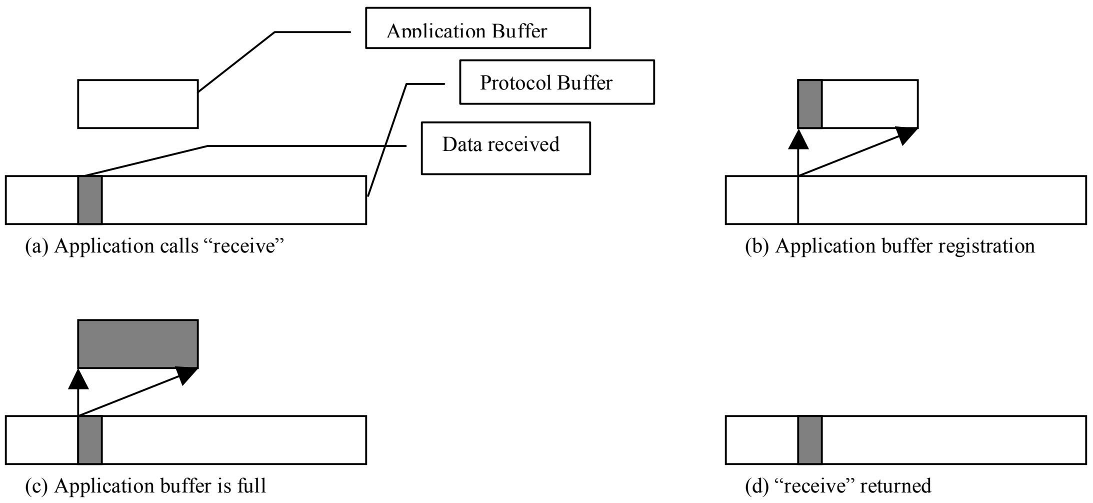

Application Buffer
Protocol Buffer
Data received
(a) Application calls "receive"
(b) Application buffer registration
(c) Application buffer is full
(d) "receive" returned

Figure 5. User Buffer Registration

User buffer registration provides a best-effort method to avoid memory copy. Only the data received between two "recv" calls needs to be copied, so if user calls the receive method continuously, the copy overhead will be very small.

3.6 Rate Control

SABUL calculates the sending rate as number of packets per second since most of the data packets are the same size. The rate control mechanism tunes the interval between two packets according the set of functions below:

$$\delta_{n+1} = \delta_n (1 + k_1(\rho - \alpha)) + c, if\ \rho > \alpha \tag{1}$$

$$\delta_{n+1} = \delta_n (1 + k_2(\rho - \alpha)), if\ \rho < \alpha \tag{2}$$

$$\delta_{n+1} = \delta_n + d, if\ \rho = \alpha \tag{3}$$

$$\delta_{n+1} = \delta_n + b, if\ L > V \tag{4}$$

where δ is the packet sending interval, ρ is the loss rate, α is the maximum tolerance loss rate, k1, k2, b, c, and d are constant parameters, L is the number of loss events since last rate tuning, and V is a valve of the number of loss events before an increase to δ is performed.

The efficiency and fairness of this rate control algorithm is discussed in [16]. SABUL can reach near optimal throughput when the loss rate is kept near to zero, and it has a better bandwidth utilization than TCP. SABUL is both self-fair and TCP friendly. Note that SABUL sends out one data packet every inter-packet interval based on the tuned sending rate, while TCP sends all the packets in the window continuously in a more aggressive way.

3.7 Timing

The data receiving algorithm in section 3.2 shows that the timers in the receiver side are not accurate timer, and there are bounded delays due to blocked data receiving. This will not affect the performance provided that the bounded delays are not too large. Accurate timers with small interval, however, may consume substantial system resource, especially the CPU clock cycles, in a high load system.

The sender side deploys accurate packet sending interval timer because the packets sending interval are very small (in microseconds), any drift of an interval will cause large error for the sending rate and hence cause a performance drop.

4. Experimental Results

Currently SABUL has been implemented on LINUX platform with all the features described in section 3. Testing have been running over three high performance network test-beds of omniNET [1], StarLight [2], SARA [3], and the long distance network between them.

We emphasize here that in the real applications the transfer speed is not only limited by the network resources, but the end systems also have great impacts on the overall throughput. Moreover, the protocol implementation [9] itself also affects the performance.

Table 1 lists the system configurations of these test-beds. We use Iperf [11] to detect the maximum UDP and TCP transfer speeds of each single connection as a comparison to the SABUL throughput. The configurations of the Iperf are the defaults. The data of the UDP bandwidth is at the situation that the loss rate is less than 0.1%. Note that these values depend on the UDP or TCP configuration, such as sending/receiving buffer size.

Table 1. System Configurations of the Test-beds

Configuration	**OmniNET**[3]	**StarLight**	**SARA**	**StarLight - SARA**
Bandwidth	4 GigE	2.5 GigE	1 GigE	10 GigE
CPU	PIII 1.0G * 2	PIII 1.0G * 2	PIII XEON 1.7G * 2	-
NIC	1GigE * 2	1GigE * 2	1GigE	-
RTT	190usec	65usec	182usec	97.8msec
UDP Throughput	860Mbps	750Mbps	780Mbps	300Mbps
TCP Throughput	880Mbps	750Mbps	898Mbps	3.1Mbps
SABUL Throughput	696Mbps	588Mbps	907Mbps	933Mbps

SABUL uses different UDP buffer size with Iperf, and this is the reason why the UDP throughput over SARA and StarLight-SARA is lower than SABUL. The throughputs of SABUL are lower than TCP over omniNET and StarLight, but are higher over the latter two networks. We found this may be caused by the

[3] omniNET is a brand new testbed for GMPLS. By the time we did the experiments, the network is still under modification for improvement. The results may change in future.

CPU limit (SABUL has more overhead to process user data than Iperf, which simply sends and receives a fixed size packet).

At IGrid 2002 (3rd International Grid Conference) we successfully reached about 2.8Gbps through 3 SABUL connections from StarLight to SARA (Figure 6). Each connection has independent end nodes but they all share the same link. It is very perspective to use SABUL to transfer data in a speed as high as 10Gbps in the near future.

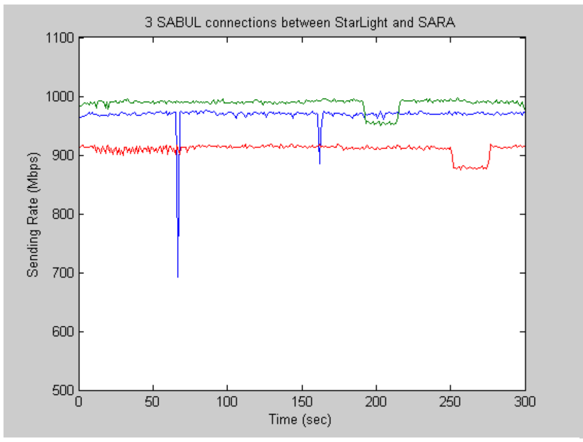


Figure 6: 3 SABUL connections between 3-macine clusters over StarLight-SARA

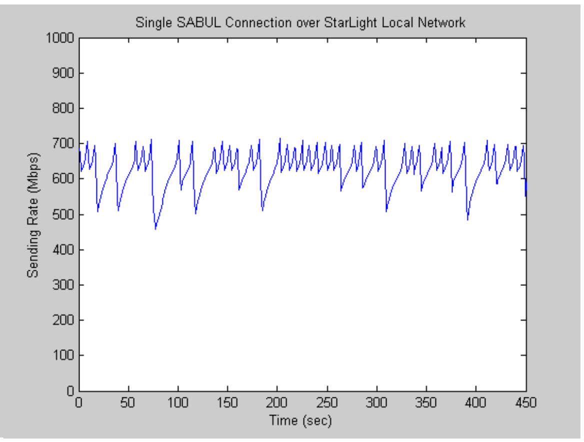


Figure 7: Single SABUL over StarLight Local Network

Figure 8 shows 2 SABUL connections with different initial rates sharing the same 1Gbps link. The figure shows that SABUL can cooperate with other SABUL connections fairly and stably. The SABUL and TCP relationship when sharing the same link is shown in Figure 9. In short RTT environments, SABUL obtains a little less bandwidth than TCP and has smaller oscillations. In long RTT environments where TCP may be limited by its window size (e.g. StarLight-SARA in Table 1), SABUL brings almost zero effect to TCP.

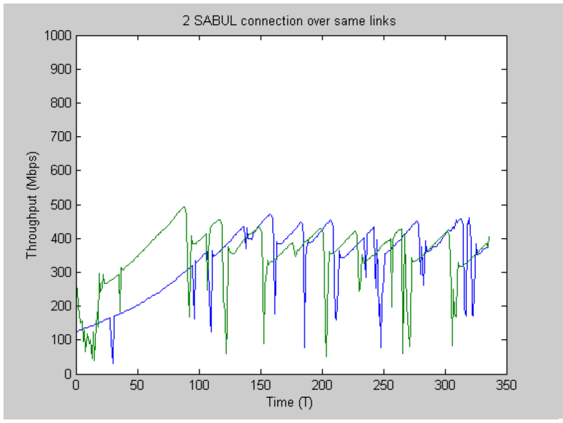


Figure 8: 2 SABUL connections over StarLight

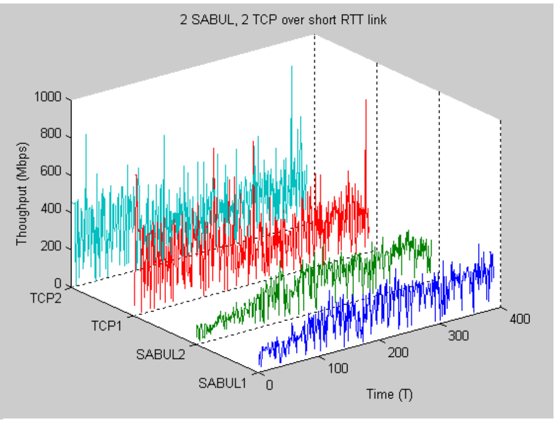


Figure 9: 2 SABUL and 2 TCP share 1Gbps on StarLight

5. Use Cases

SABUL provides similar application programming interface (API) as traditional socket interface. The sender initializes the data transfer connection and "listen" at a given port or a free port it selects, and the receiver can "connect" to the sender. Thus the SABUL connection is successfully constructed and ready for “send” and “recv” data. It also provides “sendfile” and “recvfile” functions to support file transfer directly, which is a common use scenario on high speed networks.

We have deployed SABUL in several high performance applications, including DSTP, Lambda-FTP[4], streaming data merge and Lambda Mirror.

Data Space Transfer Protocol, or DSTP, is a data service protocol for distributed data mining [14]. DSTP servers maintain the metadata information for a client to search useful data and retrieve it back to a local machine. Most of the data files in the servers are huge, such as the weather data, earth data, etc. SABUL is served as the data transfer protocol in DSTP. A typical DSTP session is: the client sends a request to construct a binary connection for data transfer with a type information for the connection type, e.g., SABUL; the server responds to the request and opens a SABUL sender, and sends the SABUL sender port number to the client; the client then opens a SABUL receiver and connects to the sender in the server side; during the following life cycle the client can send request to the server to retrieve data files and the server sends them through the SABUL connection.

Lambda-FTP uses two SABUL connections (uploading and downloading) to replace the TCP data connection in original FTP. It constructs the SABUL connection as soon as a user connects to the server, and keeps it alive until the user quits. This is not similar to the original FTP, which may open and close TCP data connection several times during the FTP session. In addition, the direct file operation support API has been used in this application – the purpose of this feature of SABUL meets the file sharing objective of FTP exactly.

The streaming data merge is another application using SABUL as its data streaming layer. The data transmission is uni-directional in the merge application: from several data sources to a single sink where the streams are merged. The data streams arrive continuously through SABUL connections and the sink machine uses a window based merge algorithm to merge the streams [15].

Lambda Mirror[5] is a network replication tool used to protect data from unexpected loss such as machine crash or network attack. The lambda Mirror cooperates with MySQL[6] database. It monitors the database file system changes and replicates the changes in a remote backup site. Any modification to the file system is copied to the backup nodes through a SABUL connection.

6. Related Work

The first protocol used to transfer bulk data over high speed networks with rate based flow control may go back to 1985, when NETBLT was proposed. NETBLT transfer fixed size bulk data and tune its sending rate after each round. The packet loss is examined in the receiver side and feedback after one block. However, because of the lack of flexibility (fixed data block size, fixed sending rate each round), the performance of NETBLT cannot be guaranteed and its utilization was limited.

Researchers have found that UDP is very suitable for multimedia data transmission. RTP/RTCP is one of the representative protocols in this field. The most important difference between such kind of protocols and the regular data transfer protocols is that multimedia protocols allow packet loss but not to retransmit them. The feedback of packet loss only leads to the degradation of source generation rate (quality of the video/audio). In addition, the rate control is done in the media source but not the network level.

Meanwhile, using UDP to transfer regular data, especially bulk data, is also becoming popular. Many works have focused on the rate control algorithms [5, 6]. Most of these algorithms are concerned with the TCP friendliness. However, these algorithms suppose the source would use a window to send packets similar to TCP. But SABUL sends out packets every fixed interval that is tuned by the rate control. The

[4] The word of “Lambda” in the project names means lambda networks, i.e. optical networks that deploy lambda technologies such as MPLS.
[5] The SABUL used in the Lambda Mirror is a little different from the protocol described in this paper. However, the basic mechanisms are similar.
[6] MySQL is an open source database product of MySQL AB Company and its contributors.

smooth flow is a "good manner" in the network since it is easy to control, and this is why SABUL is TCP friendly.

Another kind of method to overcome TCP's drawback is to use multiple TCP connections, such as PSockets [4] and GridFTP [13]. All of this kind of protocols have two major shortcomings: the aggression to the bandwidth and the protocol overhead for data processing. A parallel TCP connection with N TCP connections will obtain bandwidth resource N times as much as a coexisted traditional single TCP connection. Moreover, parallel TCP is a complex protocol, considering the complicated mechanism of TCP and the synchronization between multiple TCP connections. The complexity, hence the resource (CPU, memory, etc) cost, will become a bottleneck when the request data transfer speed is very high, because the data processing rate in the end system may not meet the network transfer rate.

Lower level solutions such as ECN and XCP [7] have appeared, but they are not to be widely adopted in the near future. Moreover, as the popularity of MPLS, such kind of mechanism may not be useful in future. Currently, an application level solution is still needed.

7. Conclusion

SABUL's advantages are that: 1) it is an application level, general purpose protocol. Applications can be immigrated between heterogeneous systems, or between high performance dedicated networks and general public networks, smoothly with small cost; 2) It supports file transfer directly, which occurs quite often in high performance applications; 3) It is not for bulk data only, i.e. there is no block size limit, and it can be used in any kind of data transfer; 4) It is TCP friendly so that we can deploy it in public networks; 5) finally, it meets the efficiency requirements for the high performance systems.

The protocol has been implemented and utilized in several applications running on Gigabit test-beds. All of them are demonstrating expected performance.

Several projects that will enhance the SABUL protocol are currently going on. We are integrating SABUL with ODIN (Optical Dynamic Intelligent Network) and THOR (Tera-Scale High Performance Optical Resource-Regulator), the middleware service modules developed by researchers in Northwestern University to provide dynamically provisioned light paths over optical links. This integration is expected to provide the high performance distributed applications a powerful data transfer layer. On the other hand, we are developing Parallel SABUL, which is a variant of SABUL but focuses on cooperating multi-processors, multiple network interface cards, and/or multi-computer clusters on the end nodes to fully utilize the extremely high bandwidth photonic links.

Reference:

[1] iCAIR, Northwestern University, *Optical Metro Network Initiative (OMNI)*, Retrieved on October 13, 2002, from http://www.icair.org/omninet/.

[2] The University of Illinois at Chicago, *StarLight*, Retrieved on October 13, 2002, from http://www.startap.net/starlight/.

[3] SARA Computing and Networking Services, the Netherlands, *SARA*, Retrieved on October 13, 2002, from http://www.sara.nl/.

[4] H. Sivakumar, S. Bailey, R. L. Grossman, *Pockets: The Case for Application-level Network Striping for Data Intensive Applications Using High Speed Wide Area Networks*, Proceedings of Supercomputing 2000, IEEE and ACM.

[5] Sally Floyd, Mark Handley, Jitendra Padhye, Jorg Winmer: *Equation-Based Congestion Control for Unicast Applications*, SIGCOMM 2001

[6] Milan Vojnovic, Jean-Yves Le Boundec: *On the Long-Run Behavior of Equation-Based Rate Control*, SIGCOMM 2002

[7] Dina Katabi, Mark Hardley, Charlie Rohrs: *Internet Congestion Control for Future High Bandwidth-Delay Product Environments*, SIGCOMM 2002

[8] Y. Gu, M. Mazzucco, X. Hong, R. L. Grossman, *Experiences on the Design and Implementation of a High Performance Protocol*, submitted to ICDCS '03.

[9] D. Clark, M. Lambert, and L. Zhang, *NETBLT: A high throughput transport protocol*, in Frontiers in Computer Communications Technology: Proc. of the ACM-SIGCOMM '87, Stowe, VT.

[10] I. Foster, C. Kesselman, S. Tuecke. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations.* International J. Supercomputer Applications, 15(3), 2001.

[11] NLANR/DAST, UIUC, *Iperf 1.6: the TCP/UDP Bandwidth Measurement Tool*, Retrieved on October 14, 2002, from http://dast.nlanr.net/Projects/Iperf/.

[12] W.T. Strayer, M.J. Lewis, and R.E. Cline, Jr., *XTP as a Transport Protocol for Distributed Parallel Processing*, Proceedings of the USENIX Symposium on High-Speed Networking, Oakland, Ca, August 1-3, 1994

[13] Globus, *GridFTP Protocol and Software*, Retrieved on October 14, 2002, from http://www.globus.org/datagrid/gridftp.html.

[14] Robert Grossman, and Marco Mazzucco, *The DataSpace Transfer Protocol (DSTP): A Data Transport Protocol for Data Webs*

[15] Marco Mazzucco, Asvin Ananthanarayan, Robert Grossman, Jorge Levera, Gokulnath Bhagavantha Rao: Merging Multiple Data Streams on Common Keys over High Performance Networks, SuperComputing 2002, Baltimore, November 2002

[16] Yunhong Gu, Xinwei Hong, Marco Mazzucco, Robert Grossman: Rate Based Congestion Control over High Bandwidth/Delay Links, Submitted to IEEE/ACM Transaction on Networking.