Experimental Analysis of the SABUL Congestion Control Algorithm*

Phoemphun Oothongsap, Yannis Viniotis, and Mladen Vouk

North Carolina State University, Raleigh NC 27606, USA

Abstract. Several new protocols such as RBUDP, User-Level UDP, Tsunami, and SABUL, have been proposed as alternatives to TCP for high-speed data transfer. The purpose of this paper is to analyze experimentally the effects of SABUL congeston control algorithm on SABUL and performance metrics such as bandwidth utilization, self-fairness, and aggressiveness. Our results confirm some expected behavior of SABUL and reveal some less expected one. Our experiments also indicate that SABUL implementation and design can result in an even more erratic behavior and degraded performance under high-congestion conditions.

1 Introduction

The high-performance networks being developed at present offer the promise of connectivity at speeds upto 40 Gbps or more. Such networks can enable new classes of high-performance applications, such as remote data analysis/visualization and high-performance grid-based computation. Although there is significant bandwidth available for such applications, the effective use of the available bandwidth is a challenge.

Several studies (e.g., [3] and references therein) have shown that, in practice, user-level distributed applications connected by a high-speed network (e.g., Abilene) cannot fully utilize the available bandwidth. The main reason for this subpar performance is the congestion control mechanisms of the transport protocol (e.g.,TCP). Thus, to improve bandwidth utilization, two alternatives are to:(i) improve the performance of TCP, and, (ii) develop new transport protocols that are suitable for a high-bandwidth environment.

An Example of new hybrid protocols is SABUL [4]. SABUL uses UDP to transfer the data and a TCP for signaling. SABUL has been evaluated empirically by simulation, and theoretically [1,4]. However, there appear be no published experiments that explore some more extreme behaviors of SABUL algorithms and implementations in a real environment. The purpose of this paper is to study SABUL behavior in situations where SABUL connections compete against each other in high congestion situations. We have also investigated how

* This work sponsored in part by the National Science Foundation grant No.NSF-9901004, DOE SciDAC grant DE-FC02-01ER25484, and IBM Corp. Shared University Research Program

SABUL send-rate calculation (and its implementation) affect SABUL performance.

The remainder of this paper is divided as follows: Section II, experimental results are presented and discussed. Section III focuses on one specific observation, that of unusually high bandwidth oscillations found in high-congestions situations.

2 Experiments

To understand the general behavior of SABUL, experiments were performed in two environments: (i) a private local area network where the round trip time (RTT) is in microseconds (a short-haul network), (ii) Abilene network where the round trip time (RTT) is in milliseconds (a long-haul network). For the long-haul network, end hosts are located at three different Abilene end-point locations: North Carolina State University (NCSU), Georgia Institute of Technology (GT) and University of Washington (UW).

The purpose of this set of experiments was to study SABUL self-fairness, bandwidth utilization, and factors affecting these properties. Multiple SABUL connections were studied in both short- and long-haul networks. We emphasize the long-haul network part because the main purpose of SABUL is to aid file transfer in high-speed long RTT networks.

Table 1 shows bandwidth utilization of three SABUL connections ($Source_1$, $Source_2$, and $Source_3$). The fourth column in Table 1 is the receiver machine. The fifth to seventh columns represent the RTT from $Source_i$ to destination. The eighth to tenth columns represent the initial sending rate of each connection in Mb/s. The eleventh to thirteenth columns represent the rate control interval (round length) of each connection in msecs. The fourteenth to sixteenth columns represent the average sending rate of each connection in Mb/s and the last column represents the figure showing the instantaneous sending rate of each experiment.

Table 1. Average sending rate of three SABUL connections

Src_1	Src_2	Src_3	Dest	RTT_1	RTT_2	RTT_3	$Init_1$	$Init_2$	$Init_3$	T_1	T_2	T_3	$Rate_1$	$Rate_2$	$Rate_3$	Figure
$n10$	n11	n12	n20	0.204	0.204	0.203	320	260	210	200	200	200	320	320	320	1
$n10$	n11	n12	n20	0.204	0.204	0.203	320	260	210	600	400	200	295	310	361	2
$fast1$	fast2	fast3	localhost	23.3	23.2	23.3	293	291	279	200	200	200	270	270	270	3
$fast1$	fast2	fast3	fasttcp	57	57.5	57	298	293	258	200	200	200	298	310	295	4
$fast1$	fast3	fasttcp	localhost	23.3	23.2	57.5	273	282	286	200	200	200	220	280	300	5

Table 1 shows the three connections compete on the same bottleneck link. The results show that SABUL connections may or may not be fair to each other. In table 1, we can categorize the experiments into three cases:(i) same RTT and rate control interval, (ii) different RTT and same rate control interval, and (iii) same RTT and different rate control interval. We notice that all connections get the similar average sending rate when rate control interval is the same regardless of RTT and initial sending rate. All connections show an unfairness behavior when the rate control interval of all connections are different. This behavior can be explained as follows. SABUL sender recalculates a new sending rate every time it receives a SYN packet from the receiver and the receiver generates a SYN packet every constant rate control interval. For the connection having a short rate control interval, the sender will receive a signal to increase a sending rate more often than the connection having a longer rate control interval. Moreover, SABUL congestion control is a variant of Multiplicative Iincrease and Multiplicative Decrease algorithm. The sender increases sending rate aggressively. Then the connection with a short rate control interval increases the sending rate more aggressively than the connection with a longer rate control interval, causing unfairness.

Figures 3 to 5 show the instantaneous sending rate of each experiment. The x-axis represents the experimental time in seconds and the y-axis represents SABUL instantaneous throughput in Mbits/sec. In each figure, we notice that SABUL still exhibits the oscillation property. And also, we notice the "synchronized" behavior, i.e., sources oscillate in phase. Synchronized behavior is an unpleasant behavior since it can reduce the overall throughput of the system. Synchronized behavior occurs due to the drop-tail operations at the router, and the round trip time (RTT) effect. With drop-tail routers, each congestion period introduces global synchronization in the network as noted in [2]. When the queues overflow, packets from several connections are dropped and these connections decrease their sending rate at the same time. The consequence is loss of throughput at the router. The effect of the RTT on the send rate fluctuation was already mentioned.

Even though it is not apparent in figures 1 and 2 due to the log scale, the behavior is still the same.

3 Discussion

In this section, we discuss some of our results in more detail. Following are some of the observations one can make from the tables and accompanying figures.

The sharp increase and decrease in the sending rate in almost every experiment seems to be a problem with SABUL implementation, and not really a defect of the SABUL congestion control algorithm. From the implementation source code, one can find that

(1) SABUL sender calculates the sending rate at the application layer.

(2) SABUL average sending rate is equal to the number of packets sent in a particular interval divided by the time interval between two SYN packets. The number of packets sent in a particular interval is calculated as follows (i) the number of retransmitted packets plus the number of new packets or (ii) if the sum of the previous value is less than the number of ERR packets, the number of packet sent is equal to number of ERR packets.

(3) At the receiver side, SABUL receiver has to estimate the average round trip time once when the program starts.

(4) SABUL receiver will send the ERR packets back to the sender in two cases: (i) once it detects the gap in packet sequence number, and (ii) periodically every 1.5 * RTT second.

These actions can cause incorrect sending rate calculation which may cause sharp increase and decrease in the measured sending rate. As we know, SABUL sender calculates the sending rate from the number of packets sent divided by the time between two SYN packets. If the number of packets sent is less than the number of packet errors in that interval, the number of packets sent is set to the number of error packets. This way of computing can cause observed effects.

(i) When the number of packets that is actually sent to the network is less than the number of error packets, then this will cause the sending rate to have a higher value than its actaul one.

(ii) SABUL receiver uses TCP channel to transmit SYN packet. SABUL sender will process SYN packets upon receipt. However, if the SABUL sender CPU is busy, then the kernel will not pass the SYN packet up to the application layer right away. With this behavior, when SABUL sender see two SYN packets back to back, the time difference between two packets is either larger than rate control interval or smaller than rate control interval.(In effect, the time measured between two SYN packets is random.)

With such unavoidable randomness and errors in the number of packets sent and the time interval between two SYN packets, SABUL sending rate will increase sharply when the sender detects a large number of losses and the time interval between two SYN packets is very small. It will also decrease sharply when the sender detects small losses (this means the actual number of packets sent is also smaller than the number of packets lost). This behavior can be seen clearly in Figure 4.

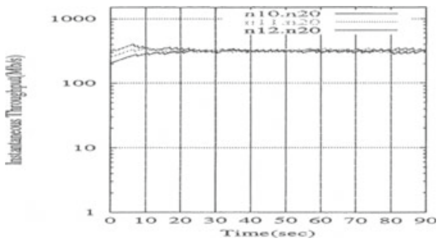

Fig. 1. Instantenous sending rate from n10, n11, and n12 to n20 with the same RTT, rate control interval and different initial sending rate

The number of error packets received from the receiver is sometimes higher than the actual number of packets lost. This phenomenon happens because the receiver periodically sends the ERR packets to the sender. When sender receives an ERR packet, it does not check whether the associated packet has already

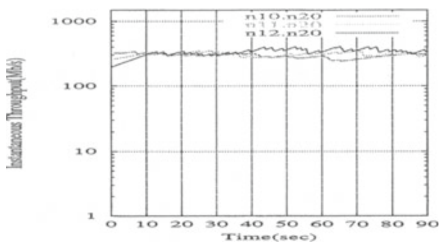

Fig. 2. Instantenous sending rate from n10, n11, and n12 to n20 with the same RTT, and different rate control interval and initial sending rate

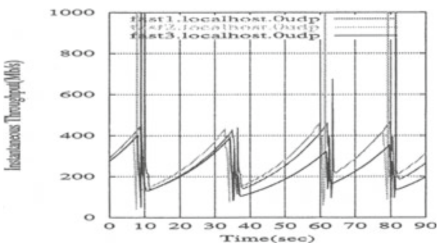

Fig. 3. Instantenous sending rate from fast1, fast2 and fast3 to localhost with the same RTT, rate control interval and initial sending rate

been retransmitted. It just adds the number of packets to the number of error packets and considers it loss total. This action, it can double the apparent error rate.

(i) the number of error packets is higher than the actual number of packets lost. This will cause a sharp increase/decrease of the sending rate as we explained previously.

(ii) the loss rate calculation is not accurate. When the number of packets sent is less than the number of error packets, SABUL sender sets the loss rate to one. This value of the loss rate will effect the new sending rate in the next interval.

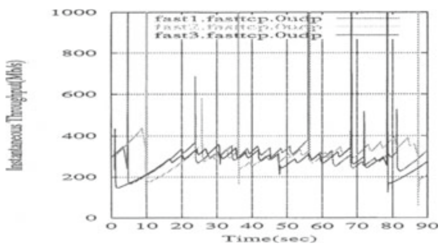

Fig. 4. Instantenous sending rate from fast1, fast2 and fast3 to fasttcp with the same RTT, rate control interval and initial sending rate

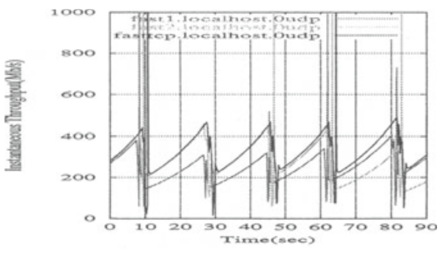

Fig. 5. Instantenous sending rate from fast1, fast3 and fasttcp to localhost with the same rate control interval, initial sending rate, and different RTT

Then in a congested network, we will see SABUL sender drop the sending rate to a smaller value. After the sender recovers all the losses, the sending rate will climb up gradually. This behavior is noticed from Figures 5.

4 Conclusion and Future Work

In this paper, we investigated performance of SABUL in both local area and wide-area network. We focused on SABUL bandwidth utilization, and SABUL self-fairness. The results of our experiments show that SABUL can utilize network bandwidth efficiently. As expected, the main factor having an effect on SABUL performance is rate control interval. SABUL shows a self-fairness property that depends on rate control interval, while RTT has no effect on SABUL self-fairness. However, we also noticed several less desirable SABUL traits: high large swing throughput oscillations (under certain conditions) and synchronization of these oscillations when several streams are involved. This degrades SABUL performance. We ascribe these behaviors not to the congestion control algorithm but to the implementation.

Acknowledgements. The authors wish to thank Dr. Steven Low and Raj Jayaram of FAST project for allowing us to use FAST project end-hosts in the experimetns, Stanilav Shalunov of Internet2 for scheduling the tests, and Scott Friedrich (of GT), Cas D'Angelo (of GT), David Richarson (of UW), Tommy Jacobson (of NCNI) for allowing us to transmit a large amount of data through their network. We would also like to thank to John Streck and John Moore of the NC ITEC and NC State Centaur Labs for advice and support given in this project, and Marhn Fullmer and Michael Bugaev for help. We would like to thank the Extreme Networks for the use of a Blackdiamond switch.

References

1. P. Oothongsap, M. Vouk, Y. Viniotis, CACC Technical Report, North Carolina State University, February 2003.

2. S. Floyd, V. Jacobson, Random Early Detection Gateways for Congestion Avoidance, *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397-413, August 1993.
3. W. Feng, P. Tinnakornsrisuphap, The Failure of TCP in High Performance Computational Grids, *Proceedings of the Super Computing 2000*, (SC2000).
4. Y. Gu, M. Mazzucco, X. Hong, R. Grossman, Rate Based Congestion Control over High Bandwidth/Delay Links, Submitted to IEEE/ACM Transaction on Networking, http://www.rgrossman.com/faq/sabul-faq-03.htm, download on February 2003.
5. M. Mazzucco, A. Ananthanarayan, R. Grossman, J. Levera, G. Rao, Merging Multiple Data Streams on Common Keys over High Performance Networks, SuperComputing 2002, November 2002.