Sync□TCP in High-Speed Environments

Ayoob Khan akhan@clemson.edu
Dhaval M. Shah dshah@clemson.edu
Zhenyu Xu zxu@clemson.edu
Clemson University, Department of Computer Science, 101 McAdams Hall, Clemson, SC 29634.

Research Advisor: Dr. Michele Weigle
Category: Graduate

1. PROBLEM AND MOTIVATION

The congestion control mechanism in TCP was first introduced by Jacobson in [7], which was later developed into TCP Tahoe. Since then, various end-to-end congestion control protocols have been proposed, including Reno [2], NewReno [6], SACK [4], and Vegas [3]. Among these protocols, TCP Reno is the standard congestion control algorithm for TCP traffic, according to [2]. However, TCP Reno detects congestion only when a packet loss occurs, i.e., when the sender receives duplicate acknowledgements (ACKs) or experiences a timeout. Hence, there are no explicit congestion notifications to end systems.

Sync-TCP [13] is a newly proposed end-to-end congestion control protocol. It is based on TCP Reno, but it can detect congestion before a packet loss occurs. This is done with the help of one-way transit times (OTTs). These OTTs provide richer congestion signals which in turn can be used to change the congestion window more effectively.

In recent years, the Internet has seen great growth in data transmission speed. However in high-speed networks, TCP congestion control is limiting the throughput [10]. There are several protocols proposed to deal with this problem, such as FAST TCP [8], High-Speed TCP [5], Scalable TCP [9], and BIC-TCP [12]. FAST TCP uses RTTs to detect congestion; however, all of the other high-speed protocols implement loss-based congestion detection.

In this paper, we present a modification to Sync-TCP which provides better throughput and performance than TCP Reno in high-speed environments. We carry out simulations in ns-2 [1] and compare our results with Reno and original Sync-TCP.

2. BACKGROUND AND RELATED WORK

Sync-TCP [11,13] is an end-to-end congestion control mechanism that makes use of one-way transit times (OTTs) in detecting network congestion. It is based on the idea of implementing the time-stamp option of TCP on the end systems. The forward path queuing delays can be determined by subtracting the minimum observed OTT from the current OTT. OTTs can determine the forward path queuing delays better than RTTs, because with RTTs there is no way to effectively distinguish if the congestion occurred in the forward path or the reverse path. Sync-TCP also compares the weighted average to a threshold based on an estimate of the maximum amount of queuing delay in the network to report the trend and where the average computed queuing delay lies. Sync-TCP traces the average computed queuing delay into one of four regions: 0-25%, 25-50%, 50-75%, and 75-100% of the maximum-observed queuing delay.

43rd ACM Southeast Conference, March 18-20, 2005, Kennesaw, GA, USA. Copyright 2004 ACM 1-58113-000-0/00/0004…$5.00.

In order to perform the trend analysis, Sync-TCP gathers nine samples of average computed queuing delays, and splits them into three groups of three samples. Then median, m_i (i = 0, 1, 2), of each of these groups is computed. The trend is determined to be increasing if $m0 < m_2$. The trend is determined to be decreasing if $m_0 > m_2$. Sync-TCP keeps computing a new trend every three new ACKs. Each time an ACK is received, Sync-TCP either reports one of eight congestion indications or that not enough samples have been gathered to compute the trend.

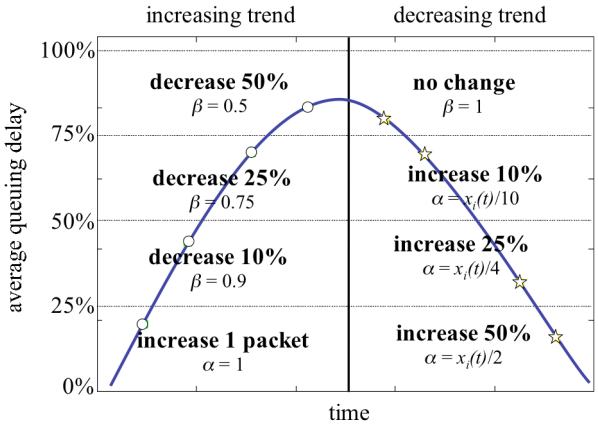

Figure 1 Sync-TCP Congestion Reactions [11]

Sync-TCP uses the AIMD congestion window adjustment algorithm with some changes in the addition (α) and multiplication (β) parameters. Sync-TCP reacts corresponding to the trend and the region of computed average queuing delay. Figure 1 shows the actions taken by Sync-TCP in response to each of the eight possible congestion indications.

3. APPROACH AND UNIQUENESS

To reduce the probability of packet loss and retain a good sending rate, we made a change to the Sync-TCP implementation.

The reason that Sync-TCP performs worse than Reno in the slow-start phase is because the maximum increment of the congestion window size is 50% of window size per RTT for Sync-TCP, whereas Reno doubles its congestion window size per RTT. To allow Sync-TCP grow its congestion window faster in slow-start phase, we let the Sync-TCP behave like Reno-TCP in slow start

until the size of congestion window reaches a pre-determined value C_{low}. Then the size of the congestion window is increased by a fixed rate of α_0, until the congestion window size reaches another value C_{high}. Once the congestion window size exceeds C_{high}, Sync-TCP resumes its original behavior, changing its congestion window size.

The constant increasing rate of congestion window size between C_{low} and C_{high} provides smooth switching from Reno behavior to Sync-TCP behavior. Without this phase, the increment of congestion window size in Reno phase will lead to abrupt changes in the queuing delays, which are used by Sync-TCP algorithm to detect congestion. By applying this modification on Sync-TCP, the performance of Reno on low-end and the performance of Sync-TCP on high-end are combined to form a congestion control that leads to better performance for the entire time period.

In our algorithm, we set C_{low}=100, C_{high}=150, and α_0=0.375 for the modified version of Sync-TCP. These parameters are chosen to balance the performance of Reno and Sync-TCP. To get better result, C_{low} should be set as high as possible as long as there is no packet loss in the Reno phase, and the C_{high} should be set such that the interval between C_{low} and C_{high} is large enough to separate the Reno phase and the Sync-TCP phase. The value of α_0 should be picked to accommodate the average increasing rate of congestion window size of Sync-TCP.

4. RESULTS AND CONTRIBUTIONS

We ran simulations with two-way TCP traffic using the ns network simulator [1] for 25 seconds. We had a single source node and destination node and two finite-buffer droptail intermediate routers. Table 1 enlists the experimental cases.

Table 1 Experimental Cases

	Bandwidth	**Propagation delay for each link**	**Maximum Congestion Window**
Low speed	10 Mbps	10 ms	50
High speed	100 Mbps	10 ms	500

For the low speed networks, the congestion window for TCP Reno took 0.4 seconds to reach the window size of 50. But Sync-TCP takes 0.85 seconds to attain the same window size. It is due to the fact that the rate of increase of the congestion window size of Sync-TCP is less when compared to that of TCP Reno. After making the above modification, Sync-TCP behaves exactly like TCP Reno, thereby giving an improved congestion window size.

For the scenario of high speed environment (shown in Figure 2), Sync-TCP reached its maximum window size in 8 seconds, when TCP Reno could not reach this size until 25 seconds. But the performance of original Sync-TCP is somewhat deteriorated due to the packet losses. If these losses could be reduced, then the throughput of Sync-TCP can be further improved. Modified Sync-TCP avoids the early packet loss and is able to achieve its maximum window size in only 4 seconds. This clearly shows that allowing Sync-TCP to behave like Reno during the slow start phase up to some threshold value leads to a great improvement in the throughput on high speed environments.

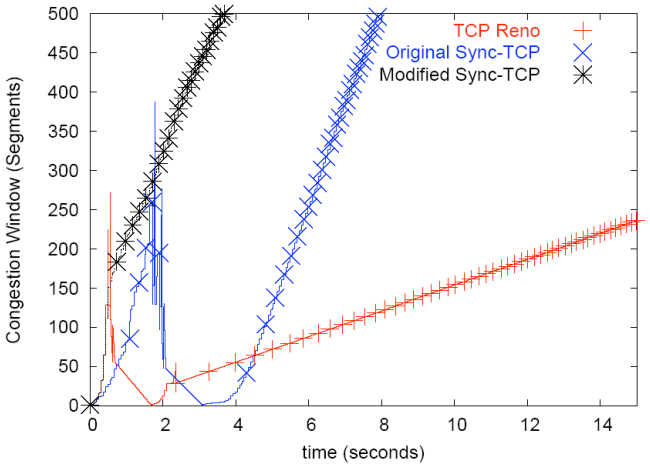

Figure 2 TCP Reno vs. Sync-TCP vs. Modified Sync-TCP

5. REFERENCES

[1] S. McCanne and S. Floyd. ns Network Simulator. http://www.isi.edu/nsnam/ns/, 2004.

[2] M. Allman, V. Paxson, and W. R. Stevens. TCP Congestion control. RFC 2581, April 1999.

[3] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: New techniques for congestion detection and avoidance. In Proceedings of ACM SIGCOMM 1994, page 2435, 1994.

[4] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno, and Sack TCP, Computer Communication Review, 26(3):5 (21, July 1996).

[5] S. Floyd. Highspeed TCP for large congestion windows. RFC 3649, December 2003.

[6] S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 2582, 1999.

[7] V. Jacobson. Congestion avoidance and control. In Proceedings of the SIGCOMM 88 Symposium, 1988.

[8] C. Jin, D. X. Wei, and S. H. Low. Fast TCP: motivation, architecture, algorithms, performance. In Proceedings of IEEE INFOCOM, March 2004.

[9] T. Kelly. Scalable TCP: Improving performance in highspeed wide area networks, 2003.

[10] J. Kurose and K. Ross. Computer Networking: A Top-Down Approach Featuring the Internet. Addison-Weasley, 3rd edition, 2004.

[11] M. C. Weigle. Investigating the Use of Synchronized Clocks in TCP Congestion Control. PhD thesis, University of North Carolina at Chapel Hill, August 2003.

[12] L. Xu, K. Harfoush, and I. Rhee. Binary increase congestion control for fast long-distance networks. In Proceedings of IEEE INFOCOM, March 2004.

[13] M.C. Weigle, K. Jeffay, and F.D. Smith, Delay-Based Early Congestion Detection and Adaptation: Impact on web performance, Computer Communications: The International Journal for the Computer and Telecommunications Industry, accepted November 2004, to appear.