TCP-Vegas Slow Start Performance in Large Bandwidth Delay Network*

Soo-hyeong Lee[1], Byung G. Kim[2], and Yanghee Choi[1]

[1] Department of Computer Science and Engineering,
Seoul National University, Seoul, Korea,
shlee@mmlab.snu.ac.kr
[2] Department of Computer Science,
University of Massachussetts at Lowell, Lowell, MA, USA

Abstract. With the rapid expansion of the Internet, it has become possible for end hosts that are separated long apart to be connected through high bandwidth links. This environment, called a Large Bandwidth Delay Network, poses a major challenge to the performance of the Internet. A long-delay connection usually suffers from being treated unfairly when competing with short-delay connections. A link, to avoid being under-utilized, has to be equipped with an a buffer as large as the bandwidth delay product of the longest connection.
TCP-Vegas is known as a potential solution to these problems. According to a number of previous studies, it is said to fairly treat connections with different propagation delays and avoid under-utilization even with a buffer that is independent of the bandwidth delay product.
In this paper we show, from simulation and analysis, that the current TCP-Vegas does NOT achieve high utilization in such a large bandwidth delay network, because of its slow-start phase. Moreover, to avoid loss, TCP-Vegas slow-start requires a buffer that is proportional to the square root of the bandwidth delay product. We propose a solution to these problems and analyze its performance.

1 Introduction

TCP is the protocol dominating the Internet[6]. TCP is a connection-oriented transport protocol that provides reliable delivery service using the unreliable IP network. There are several versions of TCP such as Tahoe, Reno, and Vegas. Every TCP version except Vegas adopts the congestion-avoidance scheme described in [1], whose basic idea is to slowly increase the number of packets in the network until a loss occurs, and to halve it when a loss does occur. Each version improves the performance of previous ones by revising behavior during loss recovery. However, Vegas tries to gain performance improvement more fundamentally, in that it does not incur loss because it does not wait for loss when reducing the number of packets in the network.

* This work was supported by Korea Science and Engineering Foundation under the contract number 20005-303-02-2. It was also supported by the Brain Korea 21 Project and National Research Laboratory Project of Korea.

TCP has two phases: congestion-avoidance and slow-start. First, we need to define two terms: window of a connection is the number of packets belonging to the connection in the network; round-trip time is the time required for a packet to traverse the network from sender to receiver and then back from receiver to sender. Slow-start doubles the window every round-trip time, whereas congestion-avoidance increases the window by a packet every round-trip time. It can be shown that slow-start effectively opens the window exponentially, whereas congestion-avoidance opens the window with a speed increase slower than linear with time.

Vegas has a new congestion detection formula[1] known as dictating the per-connection queue length in the bottleneck link[3]. Vegas applies this formula to both congestion-avoidance and slow-start to avoid loss. Vegas congestion-avoidance keeps the value of the formula within two thresholds α and β ($\alpha < \beta$): it increases the window (by one packet per round-trip time) when the formula is below α, decreases the window (by one packet per round-trip time) when the formula is above β, and keeps the window unchanged if the formula lies between the two thresholds. Vegas slow-start stops its window doubling when the value of the formula exceeds a threshold γ. The window is doubled only every second round-trip time. In between, the window stays fixed. Let us define two terms: the period of time during which the window is doubled is a *doubling round*[2], and the period of time during which the window does not change is a *holding round*. The Vegas slow-start is characterized by periodic repetition of doubling and holding.

This paper studies Vegas because it is a potential solution to be used in the Large Bandwidth Delay Network. Vegas is known to achieve high link utilization and low loss [3] [4]. In addition, [8] and [9] observed that Vegas has the virtue of having a throughput independent of propagation delay. Moreover, [8] and [13] argued that such a good performance is obtained with buffer requirement independent of propagation delay. These characteristics are substantially different to those of Tahoe and Reno. In Reno, loss is inevitable, the link utilization is highly dependent on the buffer size, throughput is a decreasing function of propagation delay, and the buffer requirement should be at least proportional to the bandwidth delay product (BDP) to achieve the same link utilization[14].

This paper studies the Vegas slow-start because it is the most important part that affects the performance of Vegas, as shown by [12]. Although it is important to study Vegas slow-start, there has been no reported research on it. Every analytical study [8][13][9] favoring Vegas has only studied the steady state congestion-avoidance phase. Their lack of study on the slow-start may be the

[1] The inventor of Vegas used the term 'congestion detection mechanism' when referring to the whole process of calculating a formula and comparing it with thresholds to decide whether or not to increase the window. By 'congestion detection formula', we mean the formula used in the congestion detection mechanism. We use this term for the purpose of discussion in this paper.

[2] *Round* is defined to be the time interval whose length is equal to the round-trip time.

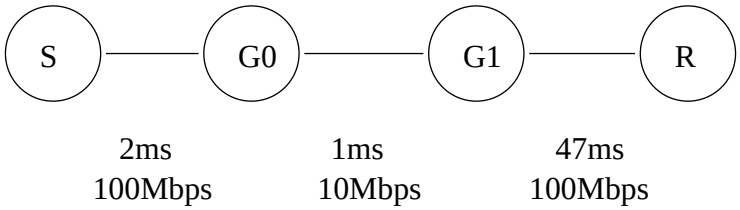

Fig. 1. Simulation Topology

reason for the discrepancy with some researchers' reports that Vegas performs poorly over either a gigabit network[5] or a long delay link[7].

This paper identifies that Vegas slow-start has two problems: 1) it is too conservative in utilizing the available bandwidth; 2) it can cause loss by overflowing the buffer. The first problem was reported recently in [12], whereas the second problem has been reported since its invention[3]. The first problem results in long convergence delay because congestion avoidance has to take the role of increasing the window. The second problem results in high maximum queue length (or equivalently, buffer requirement for no loss). This paper calculates both the convergence delay and the buffer requirement through analysis and confirms the analysis by simulation. It is worth noting that the two problems become worse as the network becomes a large bandwidth delay (LBD) network.

This paper proposes and evaluates a method to eliminate both problems. We show that a kind of pacing, named *Streak Doubling*, makes buffer requirement independent of the BDP of the connection and avoids the under-utilization.

The rest of this paper is organized as follows. We describe the problem and analyze it in Section 2. We describe our solution to the problem in Section 3. We show performance of our solution as the BDP changes in Section 4. A summary of our work and future plans is given in Section 5.

2 Symptom

The two problems, under-utilization of the link and a high buffer requirement, are apparent in the simulation. Figure 2 and Fig. 3 show the sending rate and queue length evolution of a connection under the simulation topology of Fig. 1 with a packet size of 1000B. The connection delivers data from host S to host R, through the bottleneck link G0-G1. The connection starting at 1 second under-utilizes the 10Mbps bottleneck link until it reaches the steady state at 13 seconds.

The queue length before 2.2 seconds is characterized by surges occurring every 0.2 second. Four surges are conspicuous, with heights of 2, 4, 8, and 16 packets. Given that each surge is twice as high as the previous one, the height of the largest surge would increase as the surge count increases, which is the case when the BDP increases as we show later in this section. Although the queue

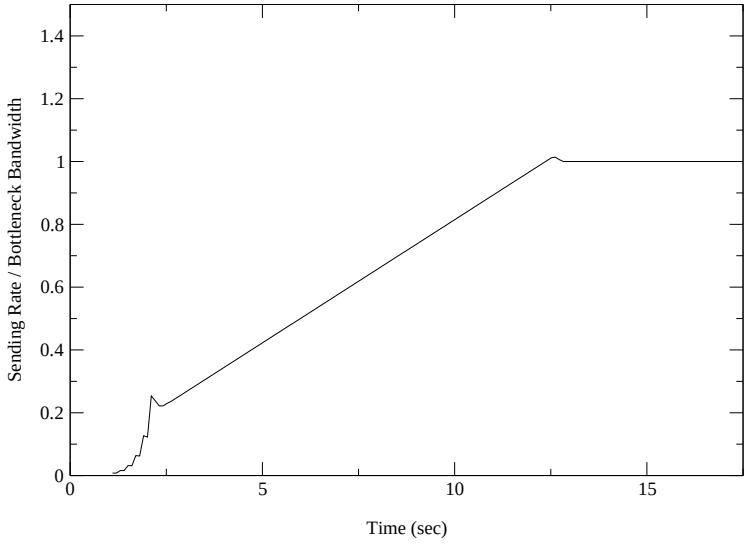

Fig. 2. Sending Rate of Normal Vegas

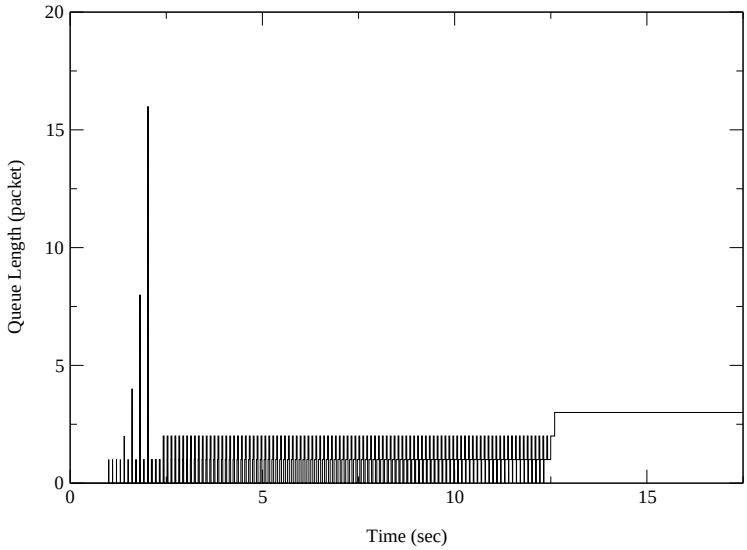

Fig. 3. Queue Length of Normal Vegas

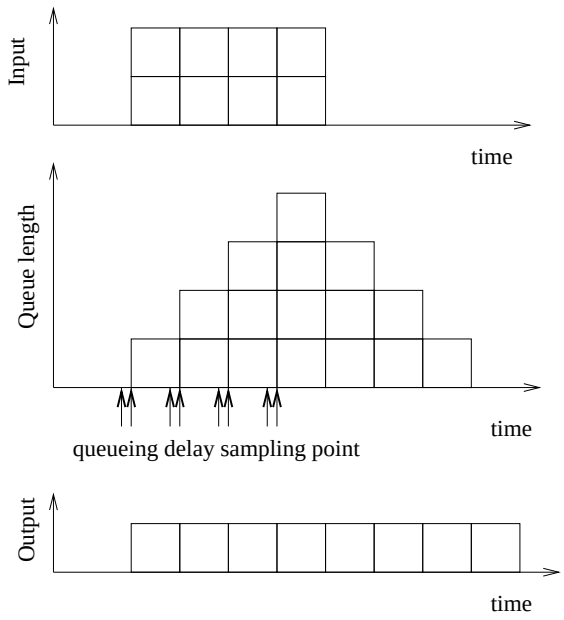

Fig. 4. Temporary Queue Buildup during Slow-Start

buildup during slow-start may not seem very high in this specific simulation scenario, we can show that it is a monotonically increasing function of BDP, whereas steady state queue length is independent of the BDP. We call the queue buildup during slow-start as *temporary* because it disappears within a round-trip time. This is observed as every surge is followed by a one-packet queue buildup in a round-trip time of 0.1 second.

Temporary queue buildup during slow-start, which is the cause of the high buffer requirement, has long been identified by researchers. It is because slow-start sends two packets on receiving an ack[3]. $W/2$ acknowledgments create W packets with sending rate double the bottleneck bandwidth, which in turn create queue length of $W/2$. The packet stream departing the bottleneck link now has a rate equal to the bottleneck bandwidth. This is depicted in Fig. 4 where $W = 8$. The horizontal unit is the packet service time and the vertical unit is packets.

The problem of under-utilizing the bandwidth is caused by a combination of temporary queue buildup and the failure of the congestion detection formula. We develop this story in the following two subsections. First, we show that the congestion detection formula of Vegas measures temporary queue buildup. Second, we show that the under-utilization becomes even worse as the BDP of the connection increases, and that buffer requirement is no exception.

[3] We use ack as an abbreviation of acknowledgment.

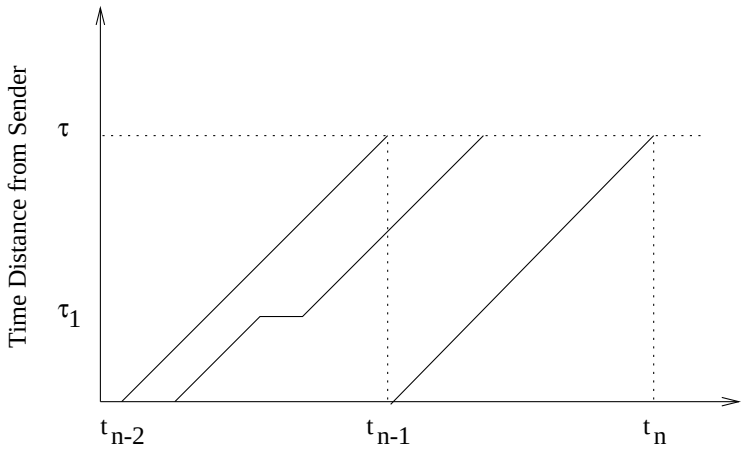

Fig. 5. Vegas Sees Queueing Delay of One and a Half RTT Before

2.1 Failure of the Vegas Congestion Detection Formula

Vegas slow-start suffers from premature termination of slow-start, because it overestimates $Diff$ due to the overestimated RTT. In this subsection, we show the reason of the RTT overestimation.

Suppose t_n is the beginning of the n-th round. The Vegas *congestion detection formula* at time t_n is as follows:

$$Diff \triangleq \left\{1 - \frac{baseRTT}{RTT(t_n)}\right\} W(t_n), \tag{1}$$

where W(t) is window size at time t and $baseRTT$ is the least of Round-Trip Times (RTT) until then. We assume $baseRTT = \tau$, where τ is the round-trip propagation delay.

Vegas slow-start checks whether $Diff > \gamma + 0.5$ only at the beginning of a doubling round. Thus t_n and t_{n-2} are beginnings of doubling rounds, and t_{n-1} is the beginning of a holding round.

$RTT(t_n)$ is the average of the round-trip times experienced by the acknowledgments received during $[t_{n-1}, t_n)$. Figure 5 shows that these are sampled by packets sent during $[t_{n-2}, t_{n-1})$. The solid line is the trajectory of a packet from when it departs the sender (the vertical axis value is 0) until it returns to the sender as an ack (vertical axis value is round-trip propagation delay τ). Through its journey, the packet cannot stop except in the bottleneck link, located τ_1 away from the sender, where it waits for service in the queue.

$RTT(t_n)$ suffers from a temporary queue buildup, because t_{n-2} is the beginning of a doubling round. Small arrows in Fig. 4 show the sampling times. Suppose C is the bottleneck link bandwidth in packets per second, then i-th packet (i begins at 1) in a round experiences a queueing delay of $\lfloor \frac{i}{2} \rfloor C^{-1}$. The

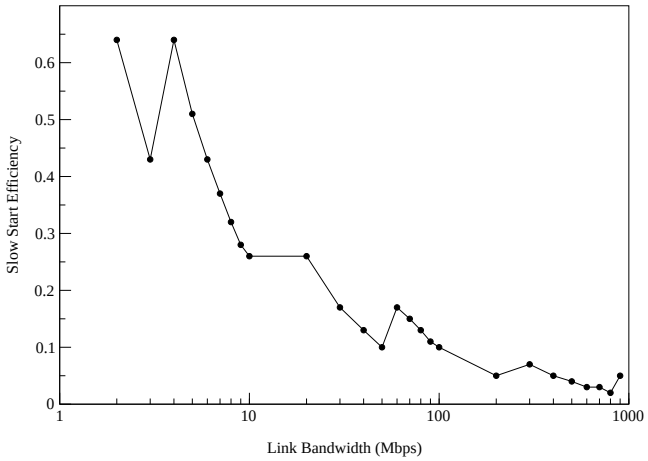

Fig. 6. Slow-Start Efficiency for Various Bottleneck Bandwidths where Round-trip Propagation Delay is 100ms and Packet Size is 1000B

average round-trip time the Vegas sender sees is

$$RTT(t_n) = baseRTT + \frac{W(t_{n-1})}{4}C^{-1}, \tag{2}$$

which is higher than the correct round-trip time, $baseRTT$.

2.2 Large Bandwidth Delay

The problem is that the termination of the Vegas slow-start does not extend in proportion to the increase of the BDP of the connection. Moreover, the rate of window increase during congestion-avoidance is indirectly proportional to BDP. Hence, as BDP increases, under-utilization of the bottleneck link becomes aggravated because the convergence delay increases.

Suppose $W^* \triangleq C\tau$ is the ideal window size that fully utilizes the bottleneck link, and the *slow-start efficiency* η is defined to be the ratio of the slow-start termination window to the ideal window. The simulation result in Fig. 6 shows that η decreases rapidly as bottleneck bandwidth increases. Increasing the propagation delay or decreasing the packet size has the same effect. In general, as we apply (2) to (1), while keeping in mind that window size assumes only integer powers of 2, we have:

$$\frac{f(\gamma_N)}{2} \leqslant \eta < f(\gamma_N) \tag{3}$$

where $\gamma_N \triangleq \frac{\gamma+0.5}{W^*}$ and $f(\gamma_N) \triangleq \gamma_N + \sqrt{16\gamma_N + \gamma_N^2}$. Note that $f(\gamma_N)$ is a strictly decreasing [4] function of W^*.

[4] η is not a strictly decreasing function of W^*. Occasional surges in Fig. 6 reflect that the window size assumes only integer powers of 2.

We have a long convergence delay if the slow-start efficiency is low and the propagation delay is high. Let us define *rise time* T_r as the portion of convergence delay other than the slow-start duration. Rise time is derived[5] as $T_r = ((1-\eta)\,W^* + \alpha)\,\tau$ if the slow-start efficiency is below unity. We can see that the rise time increases as the square of the propagation delay as we apply (3) here.

We also have a high buffer requirement even if the slow-start efficiency is low. The buffer requirement is given[6] as $Q^{max} = \frac{\eta W^*}{2}$, which increases as the square root of BDP.

One would think that controlling γ might solve some of the problems. In general, a higher γ results in a higher η, which means a lower convergence delay. However, one cannot make η fixed to a desired value by controlling γ, because the window can only assume integer powers of 2. The exact value of η is predictable but not controllable, especially when γ is relatively high compared to BDP.

There are other ways to make η closer to unity, which we show in the next section. However, high η always results in an enormous buffer requirement if two packets are sent on the moment of the receipt of an ack. In general, we face the following trade-off:

$$\frac{T_r}{\tau} + 2Q^{max} = W^* + \alpha. \tag{4}$$

As BDP increases, either round count for convergence ($\frac{T_r}{\tau}$), or the buffer requirement(Q^{max}) should be increased.

3 Solution

We consider a kind of pacing, named Streak Doubling, that strictly requires packet spacing equal to ack spacing.

We depict the operation of this algorithm and that of normal Vegas in Figs. 7 and 8. We show three consecutive doubling rounds with windows of 2, 4, and 8. The square is a packet, whose area is the amount of data that can be served by the bottleneck link during the horizontal side length of the square. Hence, the bottleneck link can serve a single row of back-to-back squares without queueing. A single column of double row packets means that the two packets are sent at the speed of the access link, which is assumed to be at least twice as fast as the bottleneck link in the figure. Normal Vegas slow-start, in Fig. 7, always sends a double row of back-to-back squares, which results in a temporary queue buildup as shown in Fig. 4. Conversely, pacing never sends a double row of back-to-back squares, as shown in Fig. 8, thus avoiding a temporary queue buildup which would lead to premature termination of the Vegas slow-start.

Streak doubling, in Fig. 8, has two periods: an active period and an inactive period. The active period has a sending rate equal to the available bandwidth,

[5] This equation is a formulation of the fact that the window increases one packet a round-trip time during the congestion-avoidance phase.

[6] This is a formulation that W packets create queue length of $W/2$ if the sending rate is double the bottleneck bandwidth. This was depicted in Fig. 4.

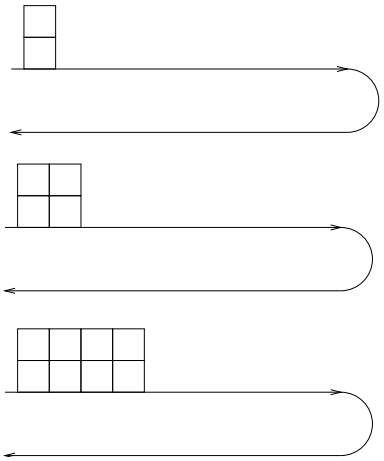

Fig. 7. Packet Spacing of Vegas slow-start

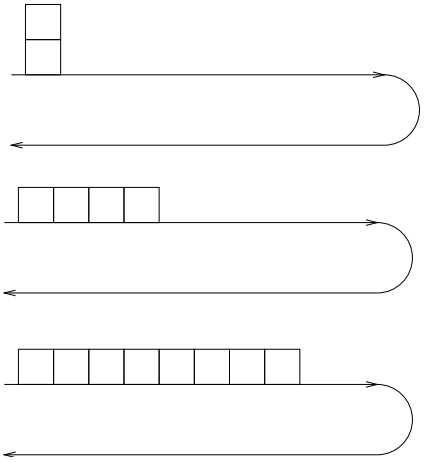

Fig. 8. Packet Spacing of Streak Doubling

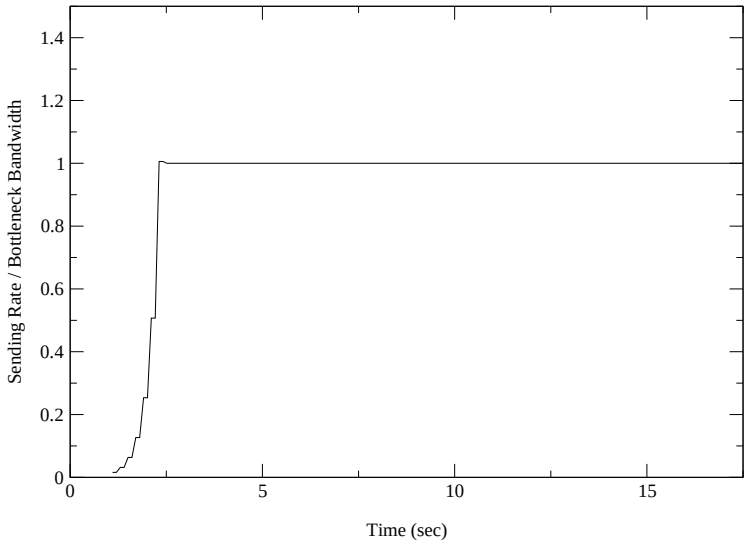

Fig. 9. Sending Rate of Streak Doubling

whereas the inactive period has a sending rate of zero. During the first half of the active period, each packet is sent on receipt of an ack without increasing the window. The latter half of the active period is a replica of the first half except that a packet is sent by a timer-scheduled event of the window increase instead of on receipt of an ack. The period of the scheduled event is the average packet service time of the bottleneck link. In the beginning, it has to send two packets as fast as possible in order to estimate the average packet service time. Streak doubling has a maximum temporary queue length of two packets because of this.

Streak doubling provides constant maximum queue length and constant round counts for rise time without regard to BDP: $Q^{\max} = 2$ and $T_r = \alpha\tau_f$. For comparison with normal Vegas slow-start, we show an equation that looks similar to (4) and is different only in the right hand side: $\frac{T_r}{\tau_f} + 2Q^{max} = 4 + \alpha$.

4 Performance

We see from Fig. 11 that the convergence delay T_c [7]of streak doubling is proportional to the propagation delay τ, whereas the convergence delay of normal Vegas increases almost as the square of the propagation delay. For a propagation delay of 1 second, streak doubling has only a one-hundredth of the convergence

[7] The convergence delay T_c for the single connection case is the sum of the rise time T_r and the slow-start duration T_{ss}. However, it is acceptable to equate convergence delay with rise time because the simulation result shows that the slow-start duration is negligible.

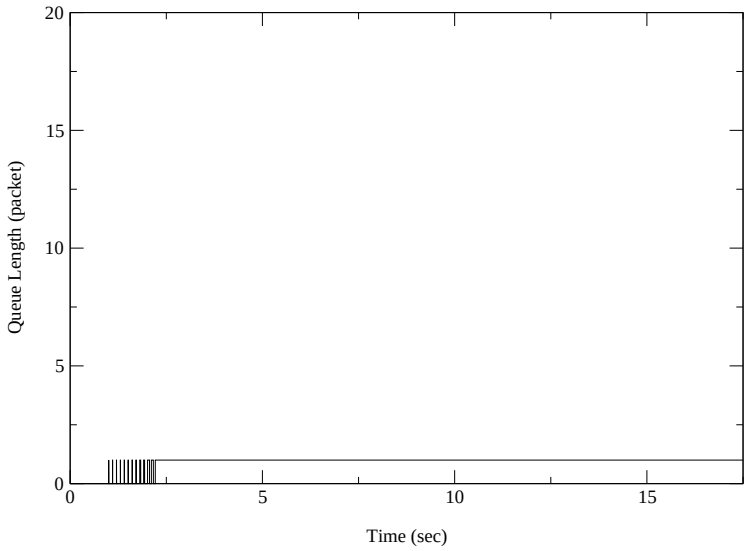

Fig. 10. Queue Length of Streak Doubling

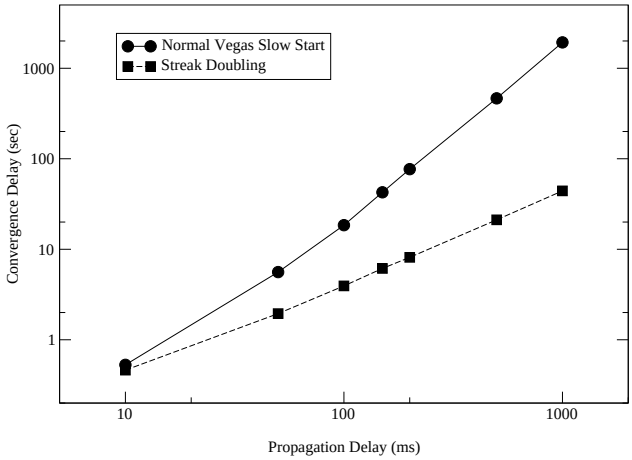

Fig. 11. Convergence Delay

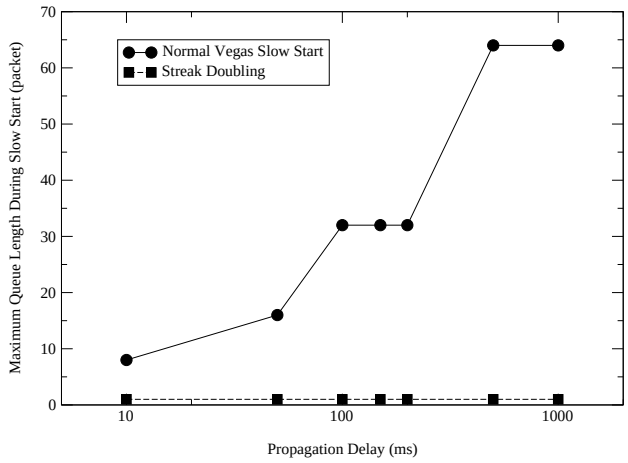

Fig. 12. Maximum Queue Length During Slow-Start

delay of normal Vegas. Using a least squared error fit, assuming a good fit to $T_c = A(\tau)^B$, we obtain the exponent of $B = 0.996$. [8] Applying the same fitting to the normal Vegas results in the exponent of $B = 1.80$.[9]

We see from Fig. 12 that the maximum queue length during slow-start is a constant of two packets for streak doubling, whereas that of normal Vegas is a monotonic increasing function of propagation delay.

5 Conclusions

Some researchers have argued that Vegas is suited for large bandwidth delay network because of its two merits in steady state: being fair irrespective of propagation delay, and highly utilizing the bandwidth with buffer requirement irrespective of bandwidth-delay product.

We have shown that the current Vegas shows poor performance in a large bandwidth delay network because of its slow-start phase. We identified two problems of Vegas slow-start: under-utilization and temporary queue buildup. The under-utilization can lead to a lower throughput of Vegas in LBD. The temporary queue buildup requires buffer proportional to the square root of the BDP. We have proposed a kind of pacing as the solution to solve the under-utilization problem with a constant buffer requirement.

We are planning two areas of further study. First, we plan to further explore whether Vegas is suitable for a large bandwidth-delay network. There are many issues left unsolved, such as convergence delay in reaching steady state in the multiple connections case. Second, we plan to implant the slow-start of Vegas

[8] τ is in $msec$ and T_c is in sec. $A = 0.040$, $\sigma[\log A] = 0.040$, and $\sigma[B] = 0.018$.
[9] $A = 0.0061$, $\sigma[\log A] = 0.150$, and $\sigma[B] = 0.0677$.

into the more popular Reno. We believe that it will improve the performance because the initial slow-start of Vegas does not incur the coarse retransmission timeout caused by burst loss. The lossless slow-start of Vegas is beneficial to its employer as well as to the network, unlike the congestion-avoidance mechanism of Vegas which is detrimental to its employer when competing with the more aggressive connections such as Reno.

References

1. V. Jacobson, *Congestion Avoidance and Control*, Proceedings of the ACM SIGCOMM'88, August 1988.
2. S. Keshav, *A Control-Theoretic Approach to Flow Control*, Proceedings of the ACM SIGCOMM '91, pp. 3-15, September 1991.
3. Lawrence S. Brakmo and Larry L. Peterson, *TCP Vegas : End to End Congestion Avoidance on a Global Internet*, IEEE J. of Selected Areas in Communication, pp.1465-1480, October 1995.
4. J. S. Ahn, P. B. Danzig, Z. Liu, and L. Yan, *Evaluation of TCP Vegas: Emulation and Experiment*, Proceedings of the ACM SIGCOMM'95, Cambridge, MA, August 1995.
5. J. S. Ahn and P. B. Danzig, *Packet Network Simulation: Speedup and Accuracy Versus Timing Granularity*, IEEE Transactions on Networking, 4(5):743-757, October 1996.
6. K. Thompson, G. Miller, and R. Wilder, *Wide-Area Internet Traffic Patterns and Characteristics*, IEEE Network, 11(6):10-23, November/December 1997.
7. Y. Zhang, E. Yan, and S.K. Dao, *A Measurement of TCP over Long-Delay Network*, Proceedings of 6th Intl. Conf. on Telecommunication Systems, pages 498-504, March 1998.
8. T. Bonald, *Comparison of TCP Reno and TCP Vegas via Fluid Approximation*, Workshop on TCP, 1998.
9. Jeonghoon Mo, Richard J. La, Venkat Anantharam, and Jean Walrand, *Analysis and Comparison of TCP Reno and Vegas*, IEEE INFOCOM'99, pp.1556-1563, 1999.
10. S. McCanne and S. Floyd, *Ns(network simulator)*, http://www-mash.cs.berkeley.edu/ns
11. G. Hasegawa, M. Murata, and H. Miyahara, *Fairness and Stability of Congestion Control Mechanisms of TCP*, IEEE INFOCOM'99, 1999.
12. U. Hengartner, J. Boliger, and T. Gross, *TCP Vegas Revisited*, IEEE INFOCOM 2000, 2000.
13. O. Ait Hellal and E. Altman, *Analysis of TCP Vegas and TCP Reno*, Telecommunications Systems, 2000.
14. H. Lee, S. Lee, and Y. Choi, *The Influence of the Large Bandwidth-Delay Product on TCP Reno, NewReno, and SACK*, the 15th International Conference on Information Networking, Beppu, Japan, January 2001.