Impact of Buffer Size on TCP Start-Up

Chadi Barakat and Eitan Altman
INRIA
2004, route des Lucioles, B.P. 93
06902 Sophia-Antipolis Cedex, France

Abstract

The Slow Start (SS) phase at the beginning of a TCP connection affects the performance of short transfers especially on long delay links such as Satellite Links. Many works have tried to improve the performance of this phase either by accelerating the congestion window increase or by setting the SS threshold at the beginning of the connection to a more accurate value in order to avoid losses and a long Timeout. However, these works don't account for the size of buffers in network nodes. In this paper, we present a general analysis of this first phase as a function of the network and TCP parameters. Among our results, we show that as claimed, the previous works improve the performance on paths with large buffers. However, on paths with small buffers, completely different results can be obtained.

1: Introduction

TCP [9, 12] is the transport protocol of the Internet. Using a congestion window (W), it controls the flow of application packets as a function of network congestion. W represents the maximum number of packets the source can transmit without the receipt of any acknowledgment (ACK) from the destination.

At the beginning of a TCP connection, a *Slow Start* phase (SS) is called to increase W quickly and smoothly [9]. It is followed by a slower increase phase called *Congestion Avoidance* (CA) [9]. The switching from SS to CA happens at a window called the SS threshold (W_{th}) which is the source estimate of the network capacity. By pipe size or by network capacity we mean in the sequel the maximum number of packets that can be fit on the path between the source and the destination.

The first SS phase is known to hurt the performance of short TCP transfers especially on long delay links such as satellite links [1, 2, 3]. This is one of the main issues discussed in the TCPSAT working group of the IETF. The first reason is that on long delay links, W needs a long time to reach large values. During this time, network resources are underutilized. The solution proposed to this problem consists in accelerating the window increase during SS. We find here Byte Counting [1] that considers the number of packets covered by an ACK while increasing the window rather than the number of ACKs. The aim is to overcome any delay ACK mechanism at the destination. We find also the Large Initial Window proposition [2] that consists in starting the connection with a window larger than 1 packet but smaller than 4 full packets.

The second problem with the first SS phase is that of losses. If W_{th} is not set appropriately at the beginning of this phase, a congestion will occur before switching to CA. Due to the fast window increase during SS, this congestion results in many losses from the same window. A long Timeout is required to recover from these losses [6]. This Timeout is followed by multiple reductions in W and W_{th} and a new SS phase. To avoid the impact of these undesirable losses on the performance, a proposition has been made in [8] to estimate a more accurate value for the SS threshold at the beginning of the connection. Normally, this threshold is set to the window advertised by the receiver [12]. The author in [8] proposes to use the flow of ACKs at the beginning of SS to estimate the Bandwidth-Delay Product (BDP) of the path and then to set W_{th} to this value.

However, these works didn't consider the problem of TCP when operating on paths with small buffers compared to their BDP. They suppose implicitly that the congestion in the network occurs during SS when the pipe size is filled. However, It is known that due to the fast window increase during SS, network buffers may overflow early before filling the pipe. With small buffers, the congestion in the network during SS is no longer an indication of the source exceeding the network capacity, but rather an indication of the network buffers failing to absorb the bursty TCP traffic during SS. This problem has been extensively studied in [4, 5, 10]. The case of a long TCP-Tahoe connection has been considered. The authors show in these works

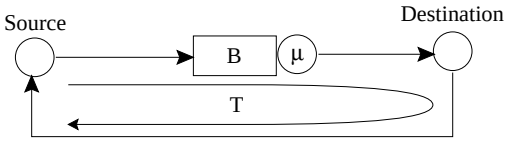

Figure 1: The network model

that small buffers result in an underestimation of network capacity and calculate the required buffer size to avoid this problem.

In this paper, we study via mathematical analysis and simulations the performance of the first SS phase of a TCP connection. We extend the model developed in [4, 5, 10] to study the first SS phase and to account for different buffer sizes and for different window increase rates. We give answers to two main questions: how to set W_{th} at the beginning of a connection and how to increase W during SS in order to avoid an early buffer overflow. Among our results, we show that the previous works may deteriorate the performance instead of improving it on paths with small buffers.

In the next section, we outline our analytical model for the evaluation of TCP performance. In section 3, we study the impact of the value given to W_{th} on TCP performance. Section 4 studies the case where W_{th} is set to a high value in such a way that losses occur during SS. The work is concluded in section 5. Our simulations are conducted with `ns`, the network simulator developed at LBNL [11].

2: A model for TCP performance

Consider a TCP connection that transfers files of size S across a path of bottleneck bandwidth μ. The widely implemented Reno version of TCP [6, 12] is used throughout the paper. However, the analysis can be applied to the other versions as well. We model the network with a single bottleneck node of rate μ and of Drop Tail buffer of size B (Figure 1). T denotes the constant component of the Round Trip Time (RTT) of the connection (the two-way propagation delay plus the service time of TCP packets in network nodes). This model has been often used in the literature to study the performance of TCP [1, 4, 5, 7, 10].

The characterization of TCP behavior during the first SS phase requires the calculation of the window at which losses occur during SS assuming that W_{th} is set to a very high value. We call this later window *the overflow window* and we denote it W_B. The other works assume implicitly that it is equal to the pipe size $(B + \mu T)$. We will show later that for a small buffer size, this window can be independent of the bottleneck bandwidth and only a function of B and the window increase rate. Note here that throughout the paper, we assume that the receiver window is set to a high value so that it doesn't limit the growth of W.

In addition to μ, T and B, the overflow window is a function of the rate at which TCP increases its window during SS. We call this rate the *aggressiveness* or the *burstiness* of SS. We model it with a factor d defined as follows.

The aggressiveness factor d
Let $W(t)$ denote the congestion window in packets at time t. We suppose that after one RTT, W is increased during SS by $W(t)/d$ packets. d can be the result of the receiver delaying ACKs and sending an ACK every d packets. $d = 1$ means that the receiver is acknowledging all the packets and $d = 2$ models the Delay ACK mechanism widely implemented in TCP receivers [12]. d can also model the loss of ACKs on the return path. It may also account for any window increase policy at the source different from that of standard TCP. An example is Byte Counting [1], where upon the receipt of an ACK, the window is increased by the number of acknowledged packets rather than by one packet as in standard TCP.

The overflow window W_B
This window is required to understand the impact of the different parameters on the performance. As in [4, 10], we divide a SS phase into mini-cycles (MC) of duration RTT. Let $W(n)$ be the number of packets transmitted during MC n. The next MC starts when the ACK for the first packet of these $W(n)$ packets reaches the source. According to our definition of the parameter d, the window size during the next MC is equal to:

$$W(n+1) = W(n) + W(n)/d = \alpha W(n), \qquad (1)$$

with $\alpha = (d+1)/d$.

Suppose that the recurrent relation (1) is valid for every $n \geq 0$. Suppose also that SS starts with a window equal to one packet. Thus,

$$W(n) = \alpha^n W(0) = \alpha^n.$$

During SS, packets leave the bottleneck in long bursts at rate μ. A burst of length $W(n)$ is served during MC n and it is followed by an idle period until the arrival of the burst of the following MC (Figure 2). This idle period between bursts disappears when the window size exceeds μT. The source transmits also packets in long bursts in response to the ACKs of packets of the previous MC. Given that the number of packets transmitted during a MC increases by a factor α, we

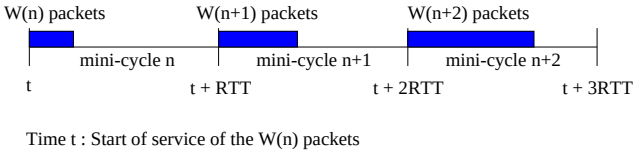

Figure 2: Bursts at the output of the bottleneck

can suppose that the long bursts transmitted by the source have an average rate $\alpha\mu$. Thus, at the beginning of MC n, the source starts to transmit a burst of length $W(n)$ at an average rate $\alpha\mu$.

When a source burst reaches the bottleneck, a queue starts to build up in B at a rate $\alpha\mu - \mu = \mu/d$. Here, two cases must be considered. The first case is when B doesn't contain any packet from the previous MC when the first packet of the burst of the current MC reaches the bottleneck. The second case is when some packets from the previous MC are still waiting in B. Note here that in the other works [4, 5, 10], only the first case has been considered. This is correct when the buffer size is very small compared to the BDP. However, the second case we introduce permits us to account for all the buffer sizes.

In the first case, a burst of size $B(d+1)$ is required to fill the buffer and causes an overflow. Let n_B^1 be the number of the MC during which B overflows. The number of packets transmitted during this MC must be larger than $B(d+1)$. But, the number of packets transmitted during the previous MC must be less than $B(d+1)$ otherwise the overflow would have occured during the previous MC. Thus, n_B^1 satisfies,

$$\alpha^{n_B^1-1} < B(d+1) \leq \alpha^{n_B^1}.$$

According to our definition of d, the transmission of a burst of $B(d+1)$ packets requires an increase in W by B packets since the beginning of MC n_B^1. It follows that,

$$W_B = W(n_B^1 - 1) = \alpha^{n_B^1-1} + B. \qquad (2)$$

Now, we consider the second case. The window size is larger than μT. The burst size required to fill the buffer is less than $B(d+1)$ since there are some packets waiting from the previous MC. It is simply equal to the number of empty places at the beginning of the MC times $(d+1)$. The increase in the window between the beginning of the MC and the overflow is equal to the number of empty places. Suppose that the overflow happens during MC n_B^2. Then, W_B becomes equal to:

$$\begin{aligned} W_B &= W(n_B^2 - 1) + B - (W(n_B^2 - 1) - \mu T) \\ &= B + \mu T. \end{aligned} \qquad (3)$$

Two expressions for W_B are then available. If the window size during MC $n_B^1 - 1$ is less than μT, then W_B will be given by equation (2), otherwise it will be given by equation (3). We can combine these two expressions into a single one as mentioned in the following theorem.

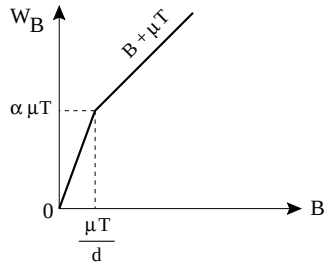

Figure 3: The overflow window W_B vs. B

Theorem 1 *If SS is not terminated before the occurrence of losses, the buffer at the entry of the bottleneck link will overflow at a window:*

$$W_B = B + \min\left(\mu T, \alpha^{n_B-1}\right),$$

with n_B given by:

$$\alpha^{n_B-1} < B(d+1) \leq \alpha^{n_B}.$$

The following corollary can be directly derived:

Corollary 1 *The bottleneck buffer will not overflow during SS if W_{th} is set less than the overflow window given by Theorem 1.*

To simplify the analysis in the sequel, we approximate α^{n_B} by $B(d+1)$. The same approximation has been made in [4, 10]. The expression of W_B becomes:

$$W_B = B + \min\left(\mu T, Bd\right). \qquad (4)$$

It is clear that this window decreases when the buffer size does. For a B less than $\mu T/d$, it becomes smaller than the pipe size and only a function of B and d. It is only for a buffer size larger than $\mu T/d$ that the pipe can be filled. For a given B and μT and while changing d, the problem of early buffer overflow during SS starts to appear once B becomes less than $\mu T/d$. The variation of W_B as a function of B is plotted in Figure 3.

3: Impact of W_{th} on the performance

In order to avoid losses and to optimize the performance, the SS threshold at the beginning of a TCP connection must be set to just less than the overflow window W_B. As we see from the expression of W_B (equation (4)), the correct value for W_{th} is a function of all the parameters not only μ and T. It decreases with the decrease in the buffer size or

Figure 4: The simulation scenario

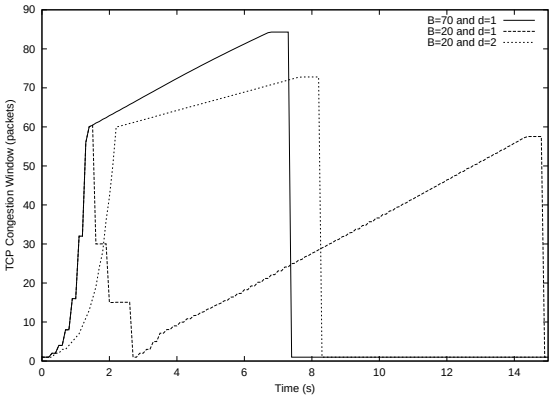

Figure 5: TCP congestion window vs. time

the increase in SS aggressiveness. If the buffer size is less than $\mu T/d$, it becomes independent of the available bandwidth! If we take as an example the value proposed for W_{th} in [8] (the Bandwidth-Delay Product). It requires a W_B larger than μT. This is possible only if $B > \mu T/(d+1)$. If ACKs are not delayed ($d = 1$), a buffer larger than half the BDP is required for this proposition to work. If ACKs are delayed, one third the BDP is required.

Consider the simulation scenario in Figure 4. TCP packets are of size 512 Bytes (without the TCP/IP header). Two buffers are considered. One of size 70 packets which is approximately equal to the BDP and the second of size 20 packets. For a 100KB file and for a $W_{th} = 50$ packets (25.6KB), we plot in Figure 5, W as function of time. We consider three cases: $B = 70$ packets and $d = 1$, $B = 20$ packets and $d = 1$, $B = 20$ and $d = 2$. For these three cases and by using (4), the overflow window is respectively equal to 140, 40 and 60 packets. According to our analysis, losses should only occur in the second case. It is clear how losses don't exist in the first case, how they appear in the second case and how they disappear in the third case.

4: Case of a high SS threshold

In this section, we study the case where W_{th} is set higher than W_B and where TCP uses its SS algorithm to gauge the network capacity. Losses occur and the performance is a function of W'_{th}, the new network capacity estimate after the recovery from losses.

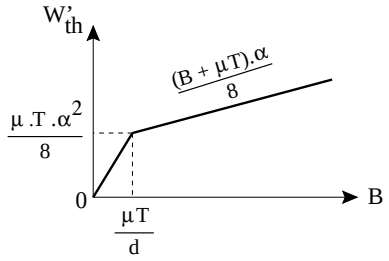

Figure 6: The new capacity estimate vs. B

4.1: Calculation of W'_{th}

The buffer overflow is detected one RTT after its occurrence. During this RTT, W increases by αW_B unless the source gets in CA. This is the case when $W_B < W_{th} < \alpha W_B$. Congestion is detected at a window W_D equal to:

$$W_D = \min\left(W_{th}, \alpha W_B\right). \tag{5}$$

Here, TCP sets W and W_{th} to half W_D and starts to recover from losses. Most often, it succeeds to detect the first two losses via Duplicate ACKs. The third and the subsequent losses need a Timeout to be detected [6]. Reno divides its window by two upon every loss detection. Thus, in most of the cases, the SS threshold after the Timeout is set to one eighth W_D. It is set to one fourth W_D when the second loss cannot be detected via Duplicate ACKs. In the sequel, we assume that $W'_{th} = W_D/8$. We assume also that W_{th} is set even higher than αW_B so that the congestion is always detected at $W_D = \alpha W_B$ (equation (5).

4.2: Impact of B and d on the performance

Using the line in Figure 3, we plot in Figure 6 the variation of W'_{th} as a function of B. We see well that the estimate provided by SS moves to zero when the buffer size does. Above $\mu T/d$, the decrease in performance is caused by the decrease in the pipe size but the aggressiveness of TCP during SS has no impact on the estimate. However, below $\mu T/d$, the buffer becomes unable to absorb the bursty traffic of TCP and the congestion appears before reaching the pipe size. An increase in TCP aggressiveness in this case reduces the estimate and may deteriorate the performance instead of improving it.

We study in the following this interaction between buffer size and TCP aggressiveness. Two cases are considered: First we suppose that the aggressiveness is controlled at the receiver which acknowledges every d data packets. In this case, both SS and CA are

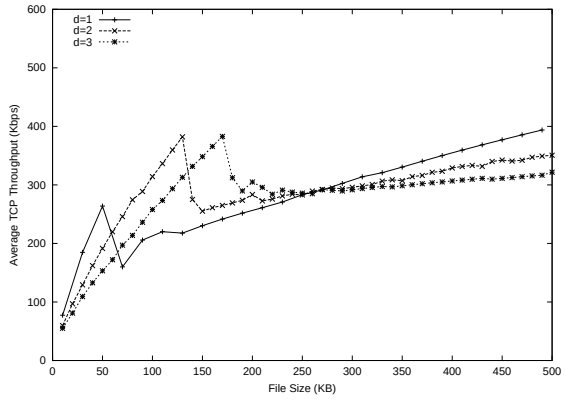

Figure 7: Throughput vs. file size for $B = 20$ packets

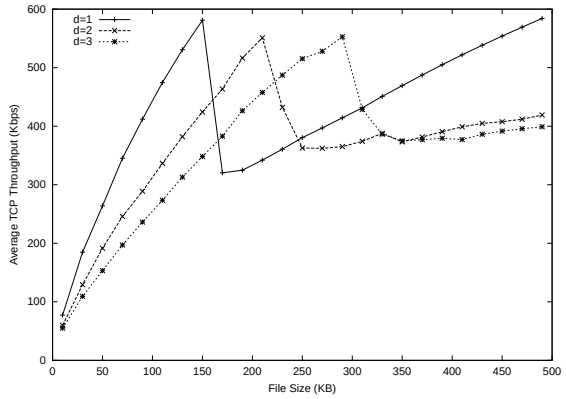

Figure 8: Throughput vs. file size for $B = 70$ packets

affected. Second, we consider the case where the aggressiveness is controlled at the source. In this case, the source can distinguish between SS and CA and can therefore adopt different factors d during each phase.

A receiver-controlled d: As we see from Figure 6, in case of a B smaller than $\mu T/d$, an increase in d improves the network capacity estimate after the Timeout. This should lead to an improvement in the performance. But, in the case of a receiver-controlled d, the increase in d slows also the window growth during CA. The gain achieved in W'_{th} when increasing d will be compensated later by the slower window growth during CA. The performance starts then to deteriorate after a small improvement.

We see this behavior in Figures 7 and 8 where we plot for two different B, 20 packets and 70 packets, the throughput as a function of the file size. Three values of d are considered. For a given d, we notice that at the beginning, the throughput increases quickly with the file size. Small files are transfered completely during the SS phase and the fast increase in the throughput is due to the fast window increase during SS.

For each d, we see also a downward jump in the performance for some file size. This corresponds to the appearance of losses during the SS phase. After this jump, the source gets in CA resulting in a slower increase in the throughput. The farther the source gets in CA, the smaller is the impact of the first SS phase. The average throughput should continue increasing in an asymptotic manner until the average throughput in the steady is reached. Normally, the maximum average throughput or the maximum utilization of network resources is a function of B, μ and T and not the aggressiveness. The aggressiveness determines the rate at which we converge to the steady state.

A small d is better than a large one whenever the file is transfered without losses. Once these losses occur for the small d (the more aggressive version), the large d starts to give better performance because it avoids the Timeout. This continues until losses occur for the large d as well. In case of small buffers, even if losses appear for the large d, it continues to give better performance than the small d for certain files sizes due to the higher network capacity estimate it provides. But, its CA phase is too slow so that it loses this privilege later. In case of large buffers however, the reduction of the aggressiveness doesn't improve the estimate. Thus, the performance of a large d becomes worse than a small d as soon as losses occur.

Thus, a receiver-controlled d affects both SS and CA. The lowest possible d must be used during CA to guarantee a fast convergence to the steady state. The most widely used factor is $d = 2$ (Delay ACK mechanism [12]). During SS, d must be chosen in such a way to increase the window as quick as possible to the fill the pipe. The best performance is obtained when we start by a small d then we switch to a larger one just before the overflow of buffer and we continue like this until we reach the pipe size. This consists in reducing SS burstiness with the increase in the congestion window. Such mechanism is difficult to implement given that TCP is not aware of the buffer size in network nodes. However, this gives us a guideline on how to change the factor d on paths with small buffers.

A sender-controlled d: We study in this section the impact of a change in the window increase rate during SS on the performance. The window increase rate during CA is kept unchanged. This permits an elimination of the impact of d on the CA phase we saw in the previous section. Such behavior requires a change at the source because it is the only element able to distinguish between the two phases. We assume in the following that the receiver

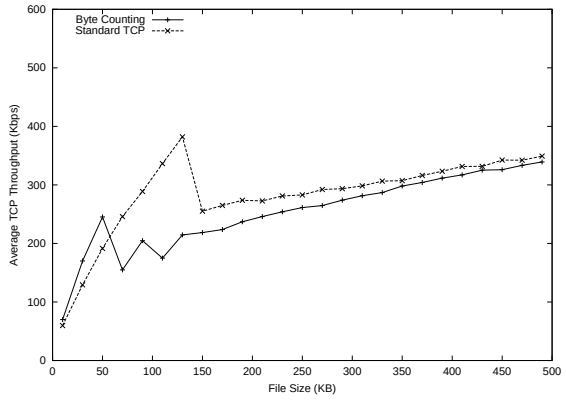

Figure 9: Standard TCP and $B = 20$ packets

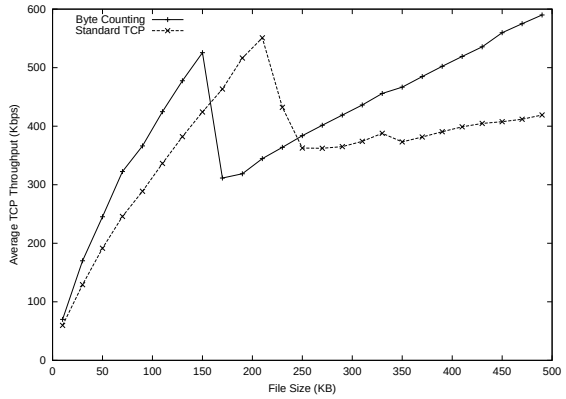

Figure 10: Byte Counting and $B = 70$ packets

is acknowledging every other data packet and that the source implements standard TCP during CA.

Changing the factor d at the source can be accomplished by changing the amount of the window increase upon the receipt of an ACK. Given that ACKs are delayed at the destination, standard TCP corresponds to $d = 2$. Byte Counting [1] corresponds to $d = 1$. Changing d from 2 to 1 has been shown to improve the performance [1, 3]. Our analysis shows that this can be the case on paths with large buffers. However, on paths with small buffers, increasing the aggressiveness reduces the network capacity estimate and deteriorates the performance.

Figures 9 and 10 explain this issue. These figures show the throughput as a function of the file size for the two buffers 20 packet and 70 packets. In each figure, one of the two lines represents standard TCP and the other line represents Byte Counting. For a large buffer, Byte Counting works perfectly and gives better performance. However, for a small buffer, Byte Counting is so aggressive that it fills the buffer before filling the pipe. This gives lower estimate and thus lower performance for most of the file sizes.

5: Conclusions

In this paper, we studied the behavior of TCP during the first SS phase. This permitted us to evaluate the performance of shorts transfers and the effectiveness of the propositions made to improve this performance. We found that, on paths with small buffers compared to their BDP, the congestion during SS may appear early before filling the pipe size. Thus, the value to give to W_{th} at the beginning of the connection in order to avoid losses must account for the buffer size otherwise losses may not be avoided. Also, in this case, we found that any increase in TCP aggressiveness deteriorates the performance instead of improving it. We presented a guideline for how to increase the window on these paths.

References

[1] M. Allman, "On the Generation and Use of TCP Acknowledgments", *Computer Communication Review*, Oct. 1998.

[2] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's Initial Window", *RFC 2414*, Sep. 1998.

[3] M. Allman, D. Glover, and L. Sanchez, "Enhancing TCP Over Satellite Channels using Standard Mechanisms", *RFC 2488*, Jan. 1999.

[4] E. Altman, J. Bolot, P. Nain, D. Elouadghiri, M. Erramdani, P. Brown, and D. Collange, "Performance Modeling of TCP/IP in a Wide-Area Network", *34th IEEE Conference on Decision and Control*, Dec. 1995.

[5] C. Barakat, N. Chaher, W. Dabbous, and E. Altman, "Improving TCP/IP over Geostationary Satellite Links", *IEEE Globecom*, Dec. 1999.

[6] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", *Computer Communication Review*, Jul. 1996.

[7] A. Kumar, "Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link", *IEEE/ACM Transactions on Networking*, Aug. 1998.

[8] J. Hoe, "Improving the Start-up Behavior of a Congestion Control Scheme for TCP", *ACM Sigcomm*, Aug. 1996.

[9] V. Jacobson, "Congestion avoidance and control", *ACM Sigcomm*, Aug. 1988.

[10] T.V. Lakshman and U. Madhow, "The performance of TCP/IP for networks with high bandwidth-delay products and random loss", *IEEE/ACM Transactions on Networking*, Jun. 1997.

[11] The LBNL Network Simulator, *ns*, http://www-nrg.ee.lbl.gov/ns.

[12] W. Stevens, "TCP Slow-Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", *RFC 2001*, Jan. 1997.