IMPROVING TCP/IP OVER GEOSTATIONARY SATELLITE LINKS

Chadi Barakat, Nesrine Chaher, Walid Dabbous and Eitan Altman
INRIA
2004, route des Lucioles, 06902 Sophia Antipolis, France

Abstract

We focus in this paper on the undesirable phenomenon of early buffer overflow during Slow Start (SS) when TCP operates in large Bandwidth-Delay Product networks such as those including Geostationary satellite links. This phenomenon, already identified in [1, 6], is caused by the bursty type of TCP traffic during SS. It results in an underestimation of the available bandwidth and a degradation in TCP throughput. Given the high cost and the scarcity of satellite links, it is of importance to find solutions to this problem. We propose two simple modifications to TCP algorithms and illustrate their effectiveness via mathematical analysis and simulations. First, we reduce the SS threshold in order to get in Congestion Avoidance before buffer overflow. Second, we space the transmission of packets during SS.

Introduction

TCP [4, 10] uses two algorithms *Slow Start (SS)* and *Congestion Avoidance (CA)* to control the flow of packets in the Internet. With SS, the Congestion Window (W) is set to one segment and it is incremented by one segment for every non-duplicate ACK received. If we suppose that the window advertised by the receiver doesn't limit the source throughput, this process continues until the SS threshold (W_{th}) is reached or losses occur. In the first case, the source moves to CA where W is increased slower by one segment for every window's worth of ACKs. In the second case, W and W_{th} are reduced and a recovery phase is called. TCP supposes that the new W_{th} is a more accurate estimate of the network capacity.

In this paper, we investigate a problem that occurs when TCP operates in a network having small buffers compared to its Bandwidth-Delay Product (BDP). It is the problem of early buffer overflow and packet losses during SS before fully utilizing the available bandwidth. These losses are due to the high rate at which TCP sends packets during SS (every ACK triggers the transmission of a burst of two packets). If the network buffers are not large enough to absorb this high rate, they will overflow early. The window size when this overflow is detected is a wrong estimation of the network capacity. But TCP considers it as the maximum reachable window and reduces its W_{th} which results in a throughput degradation. This problem has been studied in [1, 6]. In these works, the authors show that when the Tahoe version of TCP [4] is used in a network with small buffers, two consecutive SS phases are required to get in CA. A second SS is called when the buffer overflow in the first SS is detected.

Due to their high BDP and the limitations on buffer size on satellite board, this problem is very likely to appear in Geostationary satellite links. Given the high cost and the scarcity of these links, a solution to early losses during SS is required. We propose in this paper two possible changes to TCP in order to solve this problem. The impact of these changes on TCP performance is mathematically analyzed. The results are then validated by a set of simulations using `ns`, the Network Simulator [7].

Our first proposition consists in reducing the SS threshold W_{th} so that to get in CA before the overflow of buffers. A similar idea is proposed in [3] to set W_{th} to the BDP of the path crossed by the connection. In our analysis, we find the explicit expression for the required W_{th} to get rid of these early losses. A study of the throughput as a function of W_{th} is also performed.

Second, we propose to reduce the rate at which TCP transmits packets during SS. Instead of sending immediately a burst of two packets in response to an ACK, the source inserts a certain delay before the transmission of the second packet. This proposal is similar to the one proposed in [9] for spacing the ACKs on the return path. The solution in [9] requires intelligence in routers whereas our solution requires only change at the sender. Similar propositions can be found in [8, 11]. The difference from our work is that these works aim to accelerate the SS phase. They propose to bypass SS by transmitting directly at a large window which may overload the network. In our work, we keep the SS phase but we space the packets transmitted in a burst.

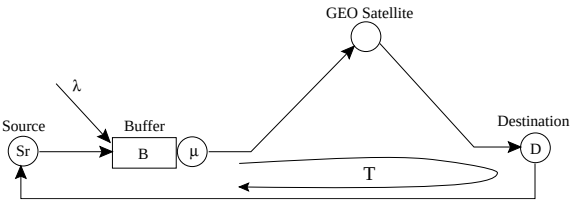

Figure 1: The Network Model

We present an analysis of the required spacing to avoid losses during SS.

The Network Model

We model the network as a single bottleneck node (N) of bandwidth μ (packets/s) and of Drop Tail buffer B of size B packets (Figure 1). TCP packets cross a GEO satellite link to the destination (D) where they are acknowledged. ACKs return to the source (S_r) via a non congested path. Let τ denote the two-way propagation delay. We suppose that S_r has always packets to send and that D acknowledges every packet. This gives us the burstiest case where S_r transmits at twice the bottleneck bandwidth. As in [1], we suppose that TCP shares the bottleneck with an uncontrolled exogenous traffic of rate λ ($\rho = \lambda/\mu < 1$ for stability requirements).

To better understand the effect of losses during SS on TCP performance, we consider the Tahoe version of TCP where SS is called upon every loss detection [4]. The other versions of TCP (Reno, New-Reno, SACK) try to avoid SS while recovering from losses [10]. However, they still resort to SS at the beginning of the connection, after a Timeout and after a long idle period. With slight modifications, our mathematical analysis can be applied to these versions as well. Even if our mathematical analysis focuses on the Tahoe version, the solutions proposed solve the problem of early losses during SS whenever this algorithm is called and independently of the version type.

Starting at a window W of one segment, the source increases W by one segment for every ACK until it reaches W_{th}. Here, it switches to CA which yields a slower increase in W by one segment every RTT. If we don't consider the transmission errors on the satellite link (i.e. by adding enough FEC), CA continues until we reach the maximum reachable window W_{max}. This is the maximum number of TCP packets that can be fit in the network. Here, a loss occurs due to buffer overflow. W_{th} is set to half W_{max}, W is set to one segment and a new SS is called. W_{max} is given by [1]:

$$W_{max} = B(\mu - \lambda)/\mu + \tau(\mu - \lambda) + 1. \quad (1)$$

We note here that the exogenous traffic reduces the share of TCP in the buffer and in the bandwidth.

This new value of W_{th} is a correct estimate of the network capacity. Network buffers must be large enough so that SS reaches this threshold without losses. What happens in the case of small buffers is that a loss occurs due to a buffer overflow before reaching W_{th}. W_{th} is then divided by two and a new SS is called. The source gets in CA at a small window and takes long time to reach W_{max}. This leads to a degradation in TCP performance.

According to [1, 6], a SS phase can be divided into mini-cycles of duration τ. At the beginning of mini-cycle n, a burst of 2^{n-1} ACKs arrives at the source at rate $\mu - \lambda/2$ and another burst of 2^n TCP packets leaves it at rate $2\mu - \lambda$. Thus, the queue at the bottleneck builds up at rate μ. The buffer overflows when we reach a certain window W_B and we are still in SS. W_B is given by [1]:

$$W_B = B(2\mu - \lambda)/\mu. \quad (2)$$

The condition to avoid losses during SS is $W_B > W_{th}$. If we define β as the normalized buffer capacity, we get the following condition found in [1]:

$$\beta = \frac{B}{\tau(\mu - \lambda) + 1} > \frac{1}{3 - \lambda/\mu}. \quad (3)$$

When the loss occurs at W_B, it isn't immediately detected. We must wait a complete RTT till the Duplicate ACKs arrive. During this time, the window grows from W_B to a value W_D which depends on the position of W_{th} with respect to W_B and $2W_B$. The maximum value of W_D is $2W_B$ but it can be equal to W_{th} if we get in CA between the overflow and its detection. Thus, we have $W_D = min(W_{th}, 2W_B)$.

The second SS is called with a threshold $W'_{th} = W_D/2$. Because this new threshold is less than W_B, we get in congestion avoidance at the end of this phase. As in the previous case, CA lasts until $W = W_{max}$.

It is clear that to avoid the losses during SS, we must maintain $W_B > W_{th}$. This can be accomplished by either increasing W_B or decreasing W_{th}. Our two propositions treat theses two possibilities.

First proposition: Decreasing the SS threshold

The idea is to decrease W_{th} to bring it below W_B. This makes the source enter CA with one SS without buffer overflow. Because W_{max} is imposed by network parameters, the only possible way is to change the reduction factor one half used by TCP in the calculation of W_{th}. When a loss is detected, we take $W_{th} = \gamma W_{max}$ with

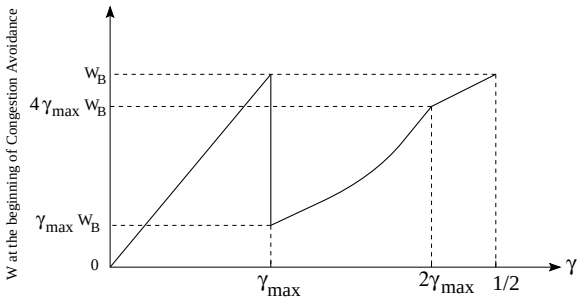

Figure 2: Model: The window at the beginning of CA

$0 < \gamma \leq 1/2$. A γ greater than $1/2$ violates the congestion control principle of TCP.

The condition to avoid losses during SS remains $W_B > W_{th} = \gamma W_{max}$. If we replace W_B and W_{max} by their values given in equations (1) and (2), we find that γ must be chosen so that

$$\gamma < \frac{\beta(2-\rho)}{\beta(1-\rho)+1} = \frac{W_B}{W_{max}} = \gamma_{max}. \qquad (4)$$

Even if it solves the problem of losses, this solution doesn't lead always to an improvement in the performance. To show this, we use the window size at the beginning of CA as a means to compare TCP performance for two different γ. In Figure 2, we show how the window at the beginning of CA varies as a function of γ. Any throughput improvement requires an increase in this window.

For a $\gamma > \gamma_{max}$, losses still exist and CA starts at W'_{th}, the threshold of the second SS phase. Moreover, the decrease in γ decreases W_{th} and then W'_{th} which deteriorates the performance instead of keeping it unchanged.

To study the effect of a $\gamma < \gamma_{max}$, we consider two cases. Such γ solves the problem of losses during SS and makes CA start at a window γW_{max}.

- $\gamma_{max} > 1/4$: The comparison must be done between $W'_{th} = W_{max}/4$ for standard TCP and γW_{max} after our modification.
- $\gamma_{max} \leq 1/4$: In this case, we must compare $W'_{th} = W_B$ for standard TCP to γW_{max} after our modification. Note that W_B is greater than $W_{max}/4$ using equation (4).

These two cases show that if the chosen factor γ is less than $1/4$, the throughput deteriorates even if losses during SS disappear. In contrast, if in the first case ($\gamma_{max} > 1/4$) we choose a factor γ between $1/4$ and γ_{max}, we get an improvement in TCP performance.

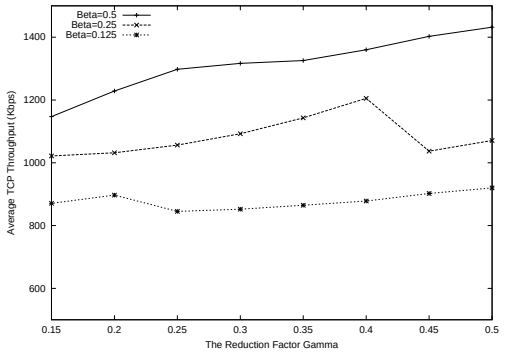

Figure 3: Simulation: TCP throughput vs. γ

This is how the reduction factor must be chosen so that our proposition works properly.

In Figure 3, we show how TCP throughput varies as a function of γ for three values of β, thus for three values of γ_{max}. The results are obtained using `ns`. The simulation scenario is described at the end of the paper. For $\beta = 0.5$, we see always a performance degradation because the problem doesn't exist ($\gamma_{max} > 1/2$). For the two other β, we notice an upward jump in the throughput for a certain γ, theoretically equal to γ_{max}. This increase is interesting for $\beta = 0.25$ given that $\gamma_{max} > 1/4$. However, for the smallest β, the fact that γ_{max} is less than $1/4$ makes impossible to find a γ that gives a better throughput than that of standard TCP ($\gamma = 0.5$).

The implementation of such solution requires changes at the source. No changes are needed at the destination. Some functions must be added at the source to estimate, on runtime, the network parameters, then the appropriate γ using equation (4). Estimation techniques as Packet-Pair [5] for bandwidth and RTT tracking [2] for buffer capacity can be adopted.

Second proposition: Packet Spacing

The idea is to keep W_{th} unchanged and to increase W_B so that to get in CA before buffer overflow. CA starts then at $W_{max}/2$ instead of $W_D/2$ yielding an improvement in TCP throughput. W_B is inversely proportional to the queue building rate. To increase it, we slow the rate at which TCP sends bursts during SS. Let R (packets/s) denote this new transmission rate. R must be taken slower than $2\mu - \lambda$, the maximum sending rate during SS. Also, R must not be less than $\mu - \lambda$, the available bandwidth at the bottleneck. An R slower than $\mu - \lambda$ makes the source the bottleneck and changes the behavior of TCP during CA which must be avoided.

To find the needed rate, we repeat the analysis in [1]

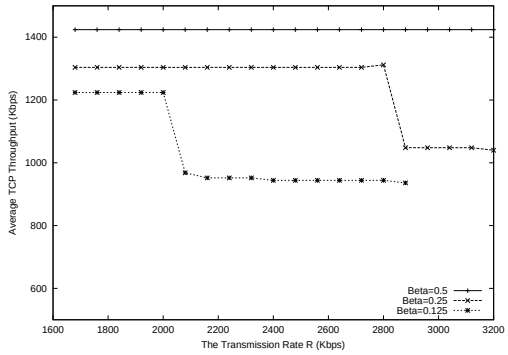

Figure 4: Simulation: TCP throughput vs. R

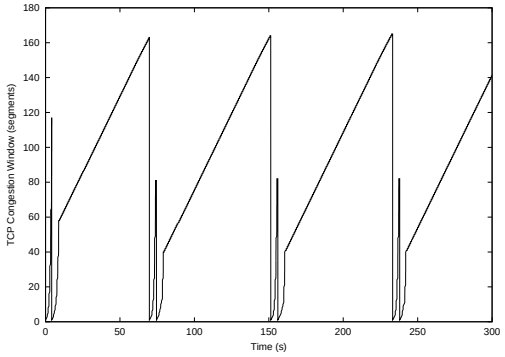

Figure 5: Simulation: Losses during SS

but now with a constant transmission rate R. Note that our proposition doesn't change the window evolution as a function of time during SS since the source is still sending a burst of two packets in response to every ACK. The difference is that a certain delay $(1/R)$ is introduced between the packets even if the output interface of the source lets it send at higher rates.

During SS, the total input rate at the bottleneck is $R+\lambda$. Thus, the queue builds up in B at rate $R+\lambda-\mu$ and TCP must keep sending packets at rate R for a time $B/(R+\lambda-\mu)$ in order to fill the buffer. As in [1], the number of packets sent during this time is taken as an approximation of the overflow window W_B. Hence,

$$W_B = BR/(R+\lambda-\mu). \tag{5}$$

We notice here that this new W_B increases when R decreases $(\lambda < \mu)$. It moves to infinity when R tends to $\mu-\lambda$ which makes losses during SS a rare phenomenon. If the problem of losses during SS exists, we can find some R between $\mu-\lambda$ and $2\mu-\lambda$ below which we have always $W_B > W_{th}$. This R is given by:

$$R < (\mu-\lambda)\frac{1+\beta(1-\rho)}{1-\beta(1+\rho)} = R_{max}. \tag{6}$$

The window growth isn't affected by the delay introduced. Thus, any R satisfying (6) results in the same performance given that the window at the beginning and at the end of CA doesn't change. This is illustrated in Figure 4 where we show the throughput for β =0.5, 0.25 and 0.125. The simulation scenario is described in the next section. Unlike the first proposition where a solution to the problem cannot be found for all the values of β, sending packets at a rate R close to $\mu-\lambda$ (1500Kbps in the figure) solves always the problem.

The implementation of this solution is easier than the first one since we don't need to follow exactly equation (6). It is enough to transmit at the available bandwidth estimate. A solution could be to transmit one TCP packet for every ACK received in a mini-cycle, then to send the remaining packets at an average rate calculated from the incoming ACKs. The advantage of this proposition compared to the ACK spacing proposed in [9] is that it does not require any intelligence in routers. It requires only minor changes at the source.

Simulations

We conduct a set of simulations using `ns` [7]. The network studied is that of Figure 1. A TCP-Tahoe connection, having an unlimited data to send and packets of total length 512bytes, is established between S_r and D. The satellite link of capacity 1.5Mbps (T1 link) and of one-way delay 250ms is crossed by a background traffic of average rate 100packets/s. TCP and background traffic packets have the same length. The TCP source is connected to the satellite link via a high speed link of rate 10Mbps and of delay 30ms. B is set to 40packets.

A simple substitution of these parameters in equation (3) shows that the problem of losses during SS exists. This is illustrated in Figure 5. TCP throughput in this case is equal to 161packets/s which represents 60.5% of the available bandwidth.

To get rid of this problem, we first reduce the factor one half according to our first proposition. Equation (4) indicates that a γ less than 0.38 must be used. Because $\gamma_{max} = 0.38 > 1/4$, a throughput improvement is possible in this scenario. Taking $\gamma = 0.34$, Figure 6 shows well the disappearance of losses. The throughput increases to 172packets/s, thus we get 4% more bandwidth utilization. In Figure 7, we divide W_{max} by more than four $(\gamma < 1/4)$. Although losses disappear, the throughput decreases to 153packets/s. In this case, it is better to keep the problem unsolved.

Now, we keep the reduction factor 1/2 unchanged and we slow the transmission. Equation (6) indicates that

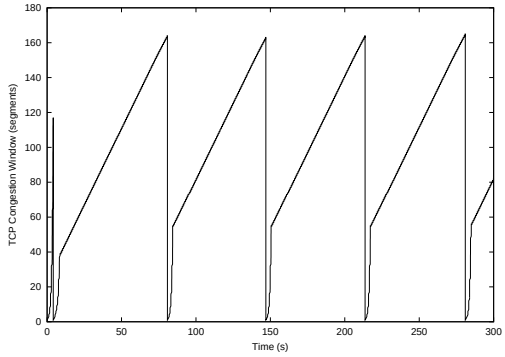

Figure 6: Simulation: $1/4 < \gamma < \gamma_{max}$

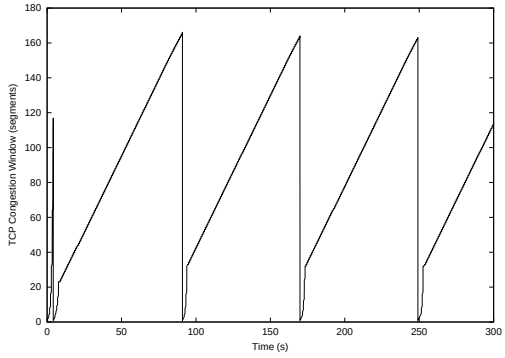

Figure 7: Simulation: $\gamma < 1/4$

we must transmit at slower than 480packets/s to solve the problem. We conducted two simulations with $R =$ 440packets/s and $R = 300$packets/s. Figures 8 and 9 show the corresponding results. We remark that these two rates solve the problem and give the same window at the beginning of CA (we must not count the first transitory TCP cycle). The throughputs are the same and are equal to 195packets/s. This gives better performance than the first proposition even for $\gamma \simeq \gamma_{max}$. Indeed, instead of starting at γW_{max}, CA starts at a large window $W_{max}/2$ after the spacing of packets.

Conclusions

In this paper, we studied the problem of early buffer overflow during Slow Start and its impact on TCP performance. Two solutions have been proposed. The first one consists in getting out Slow Start before the overflow. The second one reduces the burstiness of Slow Start by spacing the packets. The results show that the second solution is a promising one since it doesn't require a lot of information at the source and it is able to solve the problem in all the cases.

References

[1] 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.

[2] L. Brakmo and L. Peterson, "TCP Vegas: End-to-End Congestion Avoidance on a Global Internet", *IEEE Journal on Selected Areas in Communications*, Oct. 1995.

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

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

[5] S. Keshav, "A control-theoretic approach to flow control, *ACM Sigcomm*, Sept. 1991.

[6] 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.

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

[8] M. Aron and P. Druschel, "TCP: Improving Start-up Dynamics by Adaptive Timers and Congestion Control", *TR98-318*, Rice University, 1998.

[9] C. Partridge, "ACK Spacing for High Delay-Bandwidth Paths with insufficient Buffering", *Internet Draft*, Sep. 1998, Work in Progress.

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

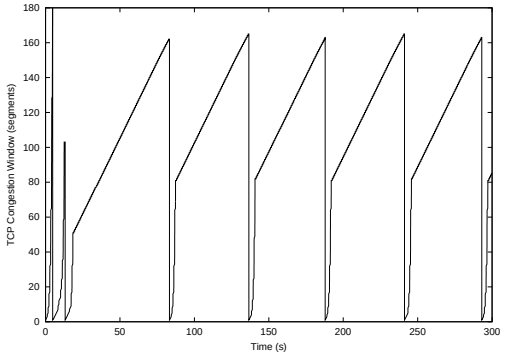

Figure 8: Simulation: $R = 440$packets/s

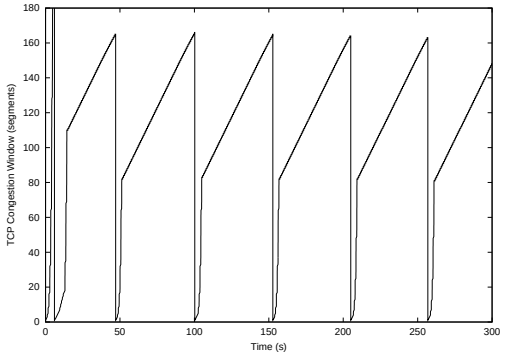

Figure 9: Simulation: $R = 300$packets/s

[11] V. Visweswaraiah and J. Heidemann, "Improving Restart of Idle TCP Connections", *Technical Report 97-661*, University of Southern California, Nov. 1997.