Performance of Short TCP Transfers

Chadi Barakat and Eitan Altman

INRIA, 2004 route des Lucioles, 06902 Sophia-Antipolis Cedex, France
{cbarakat,altman}@sophia.inria.fr

Abstract. Many works have studied the negative impact of slow start on the performance of short transfers. Some works propose to accelerate the window increase during this phase in order to improve the performance especially on satellite links. Others propose to set the slow start threshold at the beginning of the connection to a more accurate value in order to avoid losses. But, these works didn't account for the impact of network buffers on the performance. It is known that small buffers along with a fast window increase leads to an early buffer overflow and an underestimation of the network capacity. This may change completely the performance predicted by these modifications. In this paper, we present a general analysis of this first phase as a function of all the possible parameters. 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 could be obtained.

1 Introduction

TCP is the main responsible for the stability of the Internet. By varying the size of its window (W), it controls the flow of application packets so as to avoid network congestion [10]. The two main algorithms used by TCP for congestion control are Slow Start (SS) and Congestion Avoidance (CA) [10, 13]. SS is used to increase quickly W until a certain estimate of the network capacity. By network capacity or pipe size, we mean in the sequel the maximum number of packets that can be fit on the path. The network capacity estimate is called the SS threshold (W_{th}). Once W_{th} is reached, the source switches to CA where W is increased slowly to probe the network for any extra bandwidth. At any moment, the window increase is halted once the network gets congested. Congestion is detected via losses. Here, TCP sets W_{th} to half the current window, reduces its window, recovers from losses and starts again its window increase.

With SS, TCP is supposed to fill quickly the network capacity. If W_{th} is correctly set, the source switches to CA before the congestion of the network. If W_{th} overestimates the network capacity, congestion occurs during SS and W_{th} is set to a more accurate value. SS functions in this case as a quick means to gauge the network capacity. Particularly, we see this behavior of SS at the beginning of the connection where W_{th} is set to a default value usually equal to the receiver capacity (i.e. the window advertised by the receiver). But, this measurement of the network capacity is not without cost. Due to the fast window increase, SS

overloads the network and causes many losses. A long Timeout and a new SS are required to recover from these losses [7]. For this reason, a proposition to set W_{th} at the beginning of the connection to a more accurate has been made in [9]. The author proposes to set W_{th} to the Bandwidth-Delay Product (BDP) of the connection path.

TCP uses the flow of Acknowledgments (ACK) as a clock to increase W and to trigger the transmission of packets. W is increased by one packet for every ACK during SS and by one packet for every window's worth of ACKs during CA. When an ACK is received, the source transmits in a burst as many packets as its window allows. This results in a source transmission rate higher than the available bandwidth in the network when the window is increased quickly as during SS. If network buffers are not well dimensioned, losses will appear early before filling the pipe between the source and the destination or even before reaching a correctly-set SS threshold [4, 6, 11]. This results in a wrong congestion signal and an underestimation of the network capacity. This is an important problem in networks with small buffers compared to their BDP. A typical example of such networks are satellite networks where the BDP is important and where there are many limitations on the buffer size on satellite board.

The impact of buffer size on SS has been studied in many works [4, 6, 11]. These works consider a long TCP-Tahoe connection [7, 10] where SS is called after every loss detection and where W_{th} is a correct estimate of the network capacity. The aim of SS in this case is to reach W_{th} quickly and without losses. They don't consider the case where it is to SS to gauge the network capacity. They describe the problem and find an expression for the minimum buffer size required to absorb SS burstiness. However, the recent versions of TCP (Reno, SACK) [7] try to avoid SS during the steady state of the connection. The connection is supposed to stay in CA where the problem of early buffer overflow doesn't exist. We believe that the problem still exists for short transfers since TCP is obliged to start any transfer with a SS phase. Moreover, short transfers dominate most of today Internet traffic mainly due to HTTP traffic. Such transfers are in general of interactive type (e.g. Web transactions) and they are very sensitive to the service provided by the network and the underlying protocols.

In this paper, we study the impact of network buffers on the first SS phase of a TCP connection. We consider also the impact of the aggressiveness of TCP during SS. By aggressiveness or burstiness of SS, we refer in the sequel to how fast the window is increased during SS. We try to answer two main questions. First, given a certain bandwidth, a round-trip time (RTT) and a buffer size, how to set W_{th} in order to avoid losses. We show that the value to give to W_{th} is a function of all network parameters not only the BDP as suggested in [9]. Second, we consider the case where it is to SS to gauge the network capacity. We analyze in this case the effect of network parameters and the window increase rate on the capacity estimate provided by SS. We show that, for small buffers, this estimate decreases when we increase the aggressiveness of SS leading to an overall performance deterioration. This tells us that solutions like Byte Counting [1] that accelerate the window increase during SS in order to improve the performance

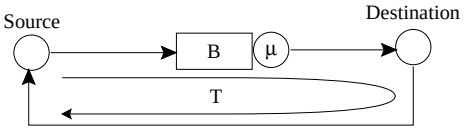

Fig. 1. The network model

may deteriorate the performance instead of improving it if network buffers are not enough large. We present guidelines for how to increase W during SS. Based on this analysis, we propose a new window increase algorithm that reduces the duration of SS while not overloading network 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}. Section 4 studies the case where W_{th} is set to a high value and where it is to SS to estimate the network capacity. In section 5, we extend our analysis to the case of multiple connections sharing the same path. Throughout the paper, analytical results are validated with a set of simulations using `ns`, the Network Simulator developed at LBNL [12]. The work is concluded in section 6.

2 A Model for TCP Performance

Consider a TCP connection that transfers files of size S. The widely used Reno version of TCP [7, 13] is used throughout the paper. However, the analysis can be applied to the other versions as well. We model the network with a single node of rate μ packets/s and of Drop Tail buffer of size B packets (Figure 1). T denotes the constant component of the RTT. This model has been often used in the literature to study the performance of TCP [1, 4, 6, 8, 11].

The performance of a short TCP transfer is strongly dependent on TCP behavior during SS. The key point in the characterization of such behavior is the calculation of the window at which losses occur during SS assuming W_{th} is set to an infinite value. We call this window *the overflow window* and we denote it by W_B. It gives us an upper bound on W_{th} since the source must get in CA before reaching W_B if it wants to avoid losses during SS. In case of losses during SS, it determines the window evolution after the recovery from losses. The new estimate of the network capacity which we denote by W'_{th} and which is equal to the SS threshold after the recovery from losses is a direct function of W_B.

Normally, W_B is equal to the pipe size $(B + \mu T)$ which has been assumed implicitly in [1, 9]. However, in [4, 6, 11], the authors have shown that W_B can take a small value in case of small buffers. However, neither the BDP nor the window increase rate has been considered. Here, we find the general expression of W_B. It is upper bounded by the pipe size and it is a decreasing function of the window increase rate during SS.

2.1 A model for TCP aggressiveness during Slow Start

Let $W(t)$ denote the window size in packets at time t. We suppose that after one RTT, the window is increased during SS by $W(t)/d$ packets. d can be the

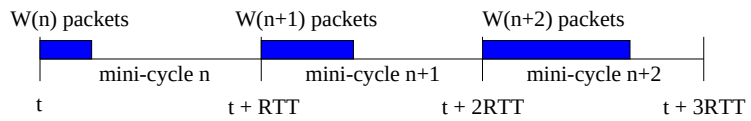

Fig. 2. Bursts at the output of the bottleneck

result of the receiver delaying ACKs and sending an ACK for every d packets and of the sender increasing its window by one packet for every non-duplicate ACK (Standard TCP [13]). $d = 1$ means that the receiver is acknowledging all the packets and $d = 2$ represents the *delay ACK* mechanism widely implemented in TCP receivers [13]. d can also account for any window increase policy at the source different than that of Standard TCP (STCP).

2.2 The Overflow Window W_B

As in [4, 11], we divide SS into mini-cycles (MC). The duration of a MC is equal to the current 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. The window size during the next MC is equal to,

$$W(n+1) = W(n) + W(n)/d = \alpha W(n), \tag{1}$$

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

We consider that STCP with non-delayed ACKs is the most aggressive case ($d \geq 1$). 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,

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

Packets are transmitted in long bursts at a rate higher than μ. They wait in B until they are served. 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 time between bursts disappears when W exceeds the BDP (μT). Given that the number of packets transmitted during a MC increases by a factor α, we can suppose that the long bursts transmitted by the source have an average rate $\alpha\mu$.

When a long source burst reaches the bottleneck, a queue starts to build up in B at a rate $\alpha\mu - \mu = \mu/d$. Two cases here 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. In [4, 6, 11], the first case has been only considered.

In the first case, a burst of size $B(d+1)$ is required to fill the buffer. 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,

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

From the first case, we have also, $\alpha^{n_B^1-1} < \mu T$. 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,

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

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 changes and becomes equal to,

$$W_B = W(n_B^2 - 1) + B - (W(n_B^2 - 1) - \mu T) = B + \mu T. \tag{3}$$

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.

Theorem 1. *If Slow Start 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 Slow Start if W_{th} is set to less than the W_B given by Theorem 1.*

To simplify the analysis, we approximate α^{n_B} by $B(d+1)$. The same approximation has been made in [4, 11]. The expression of W_B becomes,

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

It is clear that this window is equal to the pipe size $B + \mu T$ whereas B is larger than $\mu T/d$. Once B becomes less than $\mu T/d$, SS becomes unable to fill the network capacity. This is where the problem of early buffer overflow appears.

Fig. 3. The simulation scenario

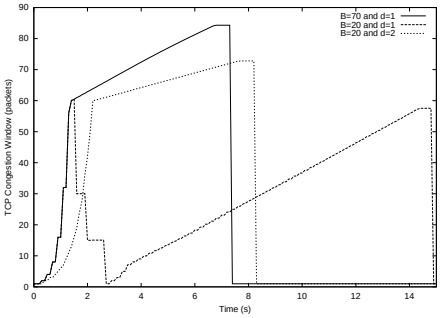

Fig. 4. TCP congestion window vs. time

3 Impact of W_{th} on the performance

The correct value for W_{th} (i.e. just less than W_B) is a function of all the parameters not only μ and T. It decreases with the decrease in the buffer size or the increase in TCP burstiness. 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 [9] (i.e. the BDP), we find that a B larger than $\mu T/(d+1)$ is required for this proposition to work otherwise losses will not be avoided.

Consider the simulation scenario in Figure 3. The source transmits a file of size 100 KB. TCP packets are of size 512 Bytes. We give two values to B, 70 packets which is approximately equal to the BDP and 20 packets. For a W_{th} equal to 50 packets, we plot in Figure 4 the congestion window as function of time. Three cases are considered, $B = 70$ packets and $d = 1$, $B = 20$ packets and $d = 1$, $B = 20$ packets and $d = 2$. d is implemented by simply delaying ACKs at the receiver. Normally, for such a threshold smaller than the BDP, one must predict that losses will not appear during SS. This is true for $B = 70$ but it is not the case for a buffer of 20 packets and a $d = 1$. A decrease in TCP burstiness is required (from $d = 1$ to $d = 2$) to help the buffer to absorb the bursts of SS.

4 Case of a High Slow Start Threshold

In this section, we study the case where TCP uses SS to gauge the network capacity. The performance is a function of W'_{th}, the capacity estimate after the recovery from losses which is a direct function of the overflow window.

4.1 Calculation of W'_{th}

The buffer overflow during SS is detected one RTT after its occurrence. During this RTT, W increases from W_B to αW_B unless the source gets in CA. This later case corresponds to $W_B < W_{th} < \alpha W_B$. Congestion detection happens 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 [7].

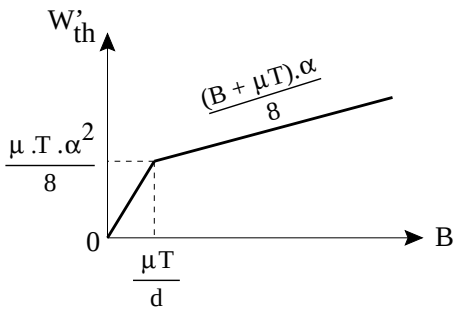

Fig. 5. New estimate vs. buffer

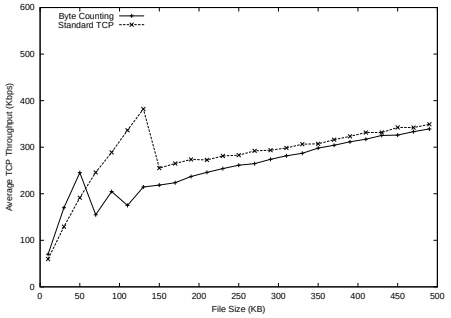

Fig. 6. BC vs. STCP for $B = 20$

However, the third and subsequent losses require a Timeout to be detected [7]. Since Reno divides its window by two upon every loss detection [7, 13], 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} is equal to $W_D/8$. Also, we suppose that W_{th} is set higher than αW_B so that the congestion is always detected at a window $W_D = \alpha W_B$.

4.2 Interaction between buffer size and SS aggressiveness

We study here the impact on the performance of the window increase rate during SS. We try to find where a given rate leads to an improvement in the performance and to define the optimum window increase strategy that works under different buffer sizes. The objective is always to reduce the duration of SS. With the increase in BDP and RTT (i.e. satellite networks), the acceleration of SS is becoming one of the main requirements for a good performance [2, 3, 5]. We assume that the window increase rate is adjusted during SS without affecting the CA phase. The easiest way to implement such kind of strategies is to work at the source since it is the only entity in the network able to distinguish between SS and CA. An example is Byte Counting (BC) [1] proposed to overcome the negative impact of the delay ACK mechanism ($d = 2$) on the performance of short transfers. Upon the receipt of an ACK, the window is increased during the first SS phase by the number of packets covered by the ACK rather than by one packet with a maximum window increase of two packets. This is equivalent to reducing d from 2 (case of STCP) to 1.

If the buffer size is large enough to absorb SS bursts, any change in d will not change the overflow window which remains equal to the pipe size. An increase in the aggressivness in this case improves the performance since it reduces the time taken by SS without changing the estimate. This happens whenever B is larger than $\mu T/d$.

The problem exists when the buffer is smaller than $\mu T/d$. As we see in Figure 5, the network capacity estimate deteriorates in this case with any increase in aggressiveness due to a deterioration of the overflow window which becomes less than the pipe size. The source gets then in CA at a small window and requires a long time to compensate the resulting capacity underestimation. It

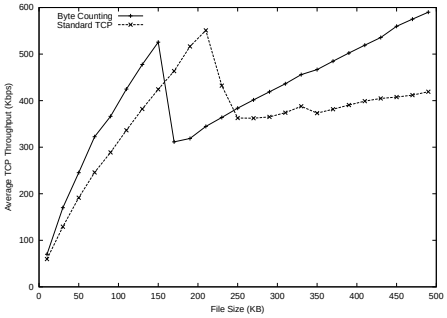

Fig. 7. BC vs. STCP for $B = 70$

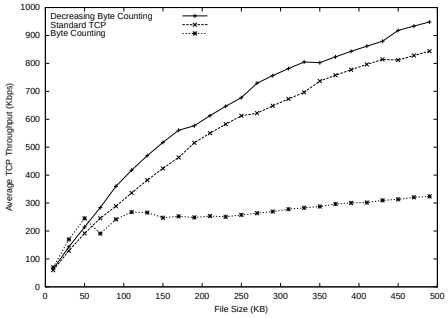

Fig. 8. Performance of DBC

is better to stop increasing the aggressiveness of SS once B becomes less than $\mu T/d$. In Figures 6 and 7, we compare BC to STCP under two buffer sizes (20 and 70 packets). The throughput is plotted as a function of the file size. For a large buffer, BC works perfectly and gives better performance. However, for a small buffer, BC is so aggressive that it fills the buffer before filling the pipe. This gives lower estimate and thus lower performance than STCP for most of the file sizes although STCP adopts a slower window increase.

The problem with BC (or other similar strategies) is that it uses the same d for the entire SS phase. However, the best performance is obtained when the source starts with a small d then switches to a larger one just before the overflow of B and it continues like this until the pipe is filled or W_{th} is reached. This consists in reducing SS burstiness with the increase in W. Such mechanism is difficult to implement given that TCP is not aware of the buffer size in network nodes. In the next section, we propose a possible solution to this problem that we call *Decreasing Byte Counting (DBC)*. It has the advantage of not requiring the buffer size but rather an idea on the ratio of the buffer size on the path to the BDP.

4.3 Decreasing Byte Counting

Fixing the same d during SS is not the optimal solution since at the beginning, the problem of aggressiveness is not as pronounced as at the end. d must be incremented gradually during SS in order to push the congestion until the overflow of the pipe. Starting at $d = 1$, d must be set to 2 when W reaches $2B$ (equation (4)), then to 3 when W reaches $3B$, then to 4 when W reaches $4B$, etc. If we consider a continuous variation of d, our analysis shows that this factor must be increased linearly with W and inversely proportional to B. There is a certain minimum limit on this factor which we fix in this work to 1. But the buffer size is not known at the source. All that the source knows is its SS threshold. Thus, instead of using B, we propose to use another factor which accounts for the the maximum value of d that it seems to be enough to reach the chosen W_{th} on a given path. Call this value d_{max}. It is a function of the ratio of the buffer size to the BDP and of the ratio of the value given to W_{th} to the BDP. If W_{th} is set at the beginning of the connection to the BDP as proposed in [9], d_{max} will be

only a function of the ratio of B to BDP. Given this d_{max}, we vary d linearly between 1 and d_{max} as long as W grows from 1 (or other initial value) to W_{th}. This gives us,

$$d(W) = 1 + (d_{max} - 1)(W - 1)/(W_{th} - 1).$$

A possible value for d_{max} can be that of STCP ($d_{max} = 2$). In this case, BC overloads the network buffers whereas STCP does not. This is equivalent to applying BC at the beginning of SS then in starting to get out of BC towards STCP as long as W grows. Our solution should give better performance than STCP and BC in this region. In the region where STCP overloads the network buffers (very small buffers), a d_{max} larger than 2 is required. Taking $d_{max} = 2$ should give poorer performance than STCP in this case but better performance than BC. Finally, one should expect that in the region where BC doesn't overload the network buffers (large buffers), BC should give the best performance.

We show in Figure 8 a comparison between BC, STCP and our proposition. W_{th} is set to the BDP. The simulation scenario is the same as that of Figure 3. A buffer size of 30 packets is taken. With such buffer, BC is unable to reach the BDP whereas STCP is. We see well how BC causes losses and reduces the performance w.r.t. STCP. Our proposition however is able to increase faster the window while avoiding losses. It provides the best performance with respect to the two others.

5 Case of Multiple TCP Connections

We describe here briefly the behavior of a new TCP connection that arrives to a network crossed by another TCP traffic. Also, we verify the benefit of our DBC algorithm in the context of many concurrent TCP connections.

5.1 A model for the case of multiple connections

It is known that in case of Drop Tail buffers and in case of close RTT, the TCP connections sharing the same bottleneck change their windows in a synchronized manner [11]. A congestion event causes losses from all connections forcing them to reduce their windows simultaneously. Suppose that all the running connections have the same RTT. Thus, the total number of packets in the network varies periodically between half the pipe size and the pipe size [11]. This behavior is independent of the number of active connections.

Suppose that a new connection arrives at a random time. Thus, it will see in the network a number of packets between half the pipe size and the pipe size. Call N the number of packets it finds. Using N, we will try to find the parameters of the equivalent single-node network seen by the new connection. Once these parameters are calculated, we can apply our previous analysis to characterize the behavior of the first SS of the new connection.

If N is smaller than μT, the new connection will see an empty buffer together with N packets propagating on the link (i.e. not waiting for service in a queue). This is because the other connections are operating in CA where no queue builds up in B until N exceeds the BDP [11]. The equivalent network seen by the new

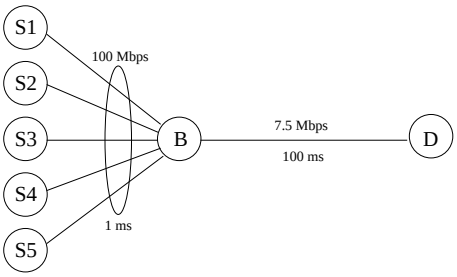

Fig. 9. The simulation scenario

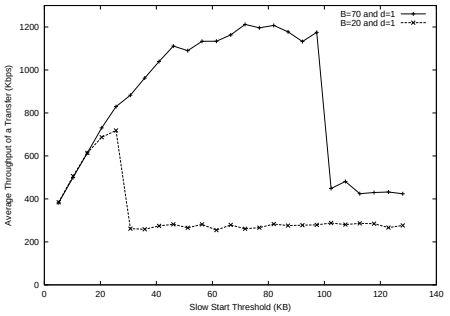

Fig. 10. Throughput vs. W_{th}

connection is formed of a buffer B and a BDP equal to $\mu T - N$. Now, if N is larger than μT, the new connection will see a full link together with $N - \mu T$ packets waiting their service in B. In this case, the equivalent network is formed only of a buffer of size $B + \mu T - N$ packets. Thus, the overflow window given in equation (4) for the single connection case can be rewritten in the case of multiple connections as,

$$W_B = B + \min\left(\mu T - N, Bd\right). \tag{6}$$

This overflow window takes its value between zero and a maximum value we call W_B^{max} which corresponds to a number of packets in the network equal to $N = (B + \mu T)/2$. W_B^{max} is equal to,

$$W_B^{max} = B + \min((\mu T - B)/2, Bd). \tag{7}$$

We see well that W_B^{max} moves to zero when B moves to zero. The performance is again an increasing function of B. But, we notice that the impact of d is less important in this case than in the case of a single connection. Indeed, in the case of multiple connections, the equivalent network seen by the new connection has the same buffer size (B) as the real network but a smaller BDP ($\mu T - N$). The real BDP is shared by the different connections whereas the buffer can be considered as dedicated to the new connection.

In the case of multiple connections, an early buffer overflow occurs when $Bd < \mu T - N$. It never occurs if the buffer size satisfies $Bd > \mu T - N$ when N is equal to $(B + \mu T)/2$. This corresponds to a buffer size larger than $\mu T/(2d + 1)$ which is less important than the buffer size required in the case of a single connection. Now, even if the buffer size is smaller than $\mu T/(2d+1)$, the problem does not always occur. It is not seen when N is close to μT.

To show this behavior in presence of multiple connections, we simulate the scenario in Figure 9. Five sources share a 7.5Mbps bottleneck link. Every source has many files to transmit. The file size is chosen randomly between 100KB and 1MB. Files of a source are transmitted on successive TCP connections. These connections are separated by a random time between 0 and 5 seconds. We run 50 simulations of 50 seconds each then we calculate the average TCP throughput during a transfer.

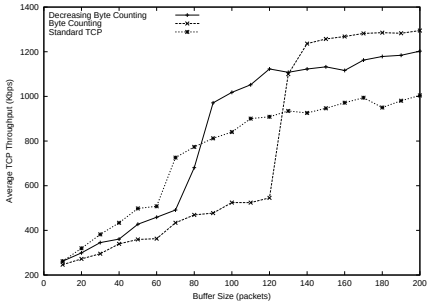

Fig. 11. Throughput vs. buffer

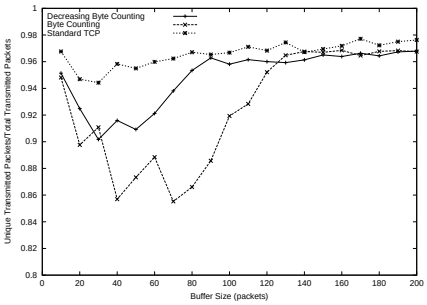

Fig. 12. The retransmission ratio

We plot in Figure 10 the throughput of a transfer as a function of W_{th}. Two cases are considered: $B = 70$ packets and $d = 1$, $B = 20$ packets and $d = 1$. The receiver sends an ACK for every other packet and the sender changes its window increase only during SS. The results in this figure match well equations (6) and (7). Theoretically, for these cases, W_B^{max} is equal respectively to 106KB and 21KB. We see well that the throughput starts to deteriorate approximately in the middle between 0 and these values of W_B^{max}. The biggest decrease in the performance appears exactly at W_B^{max}.

Thus, in case of multiple connections, W_{th} must be set according to equation (6). However, overestimating the overflow window doesn't lead to an important degradation in performance as long as W_{th} is set less than the maximum overflow window given in equation (7).

5.2 Validation of Decreasing Byte Counting

We compare here our algorithm to STCP and BC. A d_{max} equal to 2 is considered. The simulation scenario of the previous section is used. However, in this case we set W_{th} to the BDP ($\simeq 350$ packets). We change B and we plot for each strategy, two performance measures. In Figure 11, we plot the average throughput achieved by a connection. In Figure 12, we plot the average of the ratio of the number of uniquely transmitted packets to the total number of transmitted packets. We call this average the *Retransmission Ratio*. It indicates how much aggressive is a strategy. This ratio must be as much as possible close to one.

In Figure 11, we see clearly the three regions we have talked about in section 4.3. For small buffers, STCP gives the best performance but our proposition still gives better performance than BC. For a medium B, our proposition gives the best performance. At large buffers, BC is no longer aggressive and it outperforms the two other strategies but our proposition still gives better performance than STCP. Given that it merges the two strategies, the performance of DBC is either in between or better than the two others. Note here that the buffer size at which the throughputs jump up can be easily validated using equation (6).

Concerning the number of retransmitted packets, it is clear how BC gives the largest number and how STCP gives the smallest one. This number increases with the increase in the aggressiveness with an important difference between the

three strategies at small buffers. Again here, we see the three regions we talked about. The retransmission ratio of a strategy moves to one when the buffer size becomes large enough to absorb its burstiness.

6 Conclusions

In this paper, we study the behavior of SS at the beginning of a connection and its impact on the performance. We calculate first the value to which the SS threshold must be set. This value can be independent of the network capacity and function only of the buffer size. We study then the impact of the window increase rate during SS. We show that accelerating the window increase improves the performance until network buffers become unable to absorb the bursts of SS. Beyond this point, any increase in SS aggressiveness deteriorates the performance. We define the optimum window increase strategy during SS and based on this, we present a new algorithm for the window increase that reduces the duration of SS while not overloading the network buffers. With this strategy, the ACK clock is preserved and the SS threshold can be always set to the estimate of the network capacity.

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, E. Altman, and W. Dabbous, "On TCP Performance in a Heterogeneous Network : A Survey", *IEEE Communication Magazine*, Jan. 2000.
6. C. Barakat, N. Chaher, W. Dabbous, and E. Altman, "Improving TCP/IP over Geostationary Satellite Links", *IEEE Globecom*, Dec. 1999.
7. K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP", *Computer Communication Review*, Jul. 1996.
8. A. Kumar, "Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link", *IEEE/ACM Transactions on Networking*, Aug. 1998.
9. J. Hoe, "Improving the Start-up Behavior of a Congestion Control Scheme for TCP", *ACM Sigcomm*, Aug. 1996.
10. V. Jacobson, "Congestion avoidance and control", *ACM Sigcomm*, Aug. 1988.
11. 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.
12. The LBNL Network Simulator, *ns*, http://www-nrg.ee.lbl.gov/ns.
13. W. Stevens, "TCP Slow-Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", *RFC 2001*, Jan. 1997.