Round-trip Time Variation in SmoothTCP in the Face of Spurious Errors

Elvis Vieira
The University of Western Ontario/Department of Computer Science, London, Canada
Email: elvis@csd.uwo.ca

Michael Bauer
The University of Western Ontario /Department of Computer Science, London, Canada
Email: bauer@csd.uwo.ca

Abstract— **In this paper, we review the definition of a variant of TCP, called SmoothTCP, and describe one of its versions which uses ICMP-SQ messages as its primary control metric. This version of SmoothTCP is intended to be used in environments subject to spurious errors, such as in wireless networks. We evaluate the behavior of this version of SmoothTCP by comparing it with the behavior of Standard TCP in simulated environments with and without spurious errors. When there are no spurious errors, Standard TCP inherently drops packets and suffers large variations in the congestion window size causing large variations in round-trip time. In the case of spurious errors, Standard TCP encounters wide round-trip time variations around the retransmitted packet that was lost due to a spurious error. In both cases, SmoothTCP exhibits better performance with respect to round-trip time variation.**

Index Terms—**congestion control, RTT, spurious errors, TCP performance, SmoothTCP, wireless performance.**

I. Introduction

Previous work has demonstrated the existence of design problems in congestion control of Standard TCP[1] related to its use of packet drops to signal congestion[2,3,4,5,6]. Indeed, packets drops should indicate to the TCP sender a router queue, somewhere between the sender and the receiver, is overloaded. Therefore, the sender should decrease its sending rate in order to return the queue to a normal size. But this is not always true. A packet can be dropped for reasons other than a full queue. An example is what happens in wireless networks where the movement of the mobile node, the properties of the transmission media, or other equipment introducing noise, can introduce undesirable delays or spurious errors in communication. Specifically, spurious errors cause the sender to unnecessarily decrease its sending rate which also increases round-trip time (RTT) variation. In such situations, however, there is no reason to decrease the sending rate since no router queue is actually overloaded. Consequently, the sender is penalized unnecessarily because of this ***signal uncertainty***. Additionally, using packet drops as the main congestion indication creates another problem. Once a packet is dropped in a queue, the sender does not know immediately, but it must deduct by not receiving any acknowledgement for the dropped packet. This takes approximately one RTT, the time to send a packet and receive its echo, which is a ***temporal gap*** too large.

Our work suggests that these problems can be addressed by introducing the concept of ***smoothness***. Briefly, a congestion control mechanism, which has this smoothness characteristic, should react to congestion signals in a less "harsh" way than just halving its window size, as does Standard TCP. This is because, if the congestion control mechanism is "smooth", then there is almost no signal uncertainty. Specifically, this paper proposes an instance of such mechanism, using only ICMP-SQ messages to signal congestion. By using ICMP-SQ messages, the uncertainty problem is practically eliminated as long as an ICMP-SQ message is sent only when the router queue is getting overloaded and not when a spurious error happens. This can be done by setting a queue size threshold and having ICMP-SQ messages sent directly to the sender when that threshold is reached.

The remainder of this paper is organized as follows. Section II discusses the motivation behind this research and some related work. Section III introduces the definition of Smooth Congestion Control Algorithms and describes the smoothness property and the general format of their congestion window functions. Section IV describes SmoothTCP which is an instance of a Smooth Congestion Control Algorithm specifically defined to deal with signal uncertainty and temporal gap problems. In Section V, we present the RTT variation behavior of SmoothTCP in normal networks, that is, not subjected to spurious errors. In Section VI, we describe the RTT variation behavior of SmoothTCP when spurious errors are introduced in those networks. Finally, in Section VII, we summarize some conclusions about the work that has been done so far and presented in this paper.

II. Motivations and Related Work

The work in [2] suggests that the threat to the stability of the Internet originates not from flows having

alternative congestion controls, but from those not having any congestion control at all, such as large-scale multicasting flows or some real-time traffic. The stability of the Internet, therefore, does not require that flows decrease their sending rate by halving their window sizes, as in Standard TCP [7]. In particular, to avoid congestion collapse it is necessary to use a lower sending rate only when there is a high loss rate.

Two key issues related to the TCP performance can be identified under links subjected to errors: its inability to separate the packet losses due to congestion from those because of other problems; and its reliance on the timer to recover from a failed retransmission cycle. One of the effects of the issues cited above is spurious retransmission timeouts, which reduce the performance of TCP as they can start unnecessary retransmissions of segments. These problems have been noted and addressed by others:

1. Eifel Algorithm: In [3], Ludwig and Katz proposed an algorithm, called Eifel, to deal with spurious retransmission timeouts. They proposed an alteration in TCP that uses either timestamps sent in TCP segments or two of the reserved bits in the TCP header to determine whether there was any packet loss due to some error when an ACK arrived in the sender.
2. Westwood TCP: In [4] and [5] Westwood TCP is described. Instead of dropping the window size by half of the value it was before a packet loss, it proposed a linear decreasing of the congestion window depending on the bandwidth measured on each ACK arrival. This results in a performance of up to four times that of the Standard TCP when spurious errors occur.
3. F-RTO: The F-RTO algorithm is described in [6]. F-RTO only affects the TCP sender when there is a retransmission timeout. Otherwise it behaves as Standard TCP.

A TCP-friendly mechanism is described in [2] to control unicast traffic using equation-based congestion control. This mechanism avoids reducing the sending rate to react to a single packet drop. Instead, the sender adjusts its sending rate according to the measured rate of loss events in a single round-trip time. The Family of Binomial Congestion Control Algorithms [8] also looks at the generalization of congestion control; the work in [8] was the basis for the congestion control functions in the present work.

III. Smooth Congestion Control Algorithms

The basic idea of the algorithms in previous work [9,10,11] is to make changes in the congestion window as small as possible. This is the basis for our smoothness property [10], the main requirement for Smooth Congestion Control Algorithms.

A. Smoothness

A congestion control mechanism will be said to have ***smoothness characteristics*** if it has the following five properties: a smooth curve; vertical smoothness; horizontal smoothness; proactive smoothness; and precision. These are defined as follows:

- **Smooth Curve Property:** An important characteristic of TCP's standard congestion control algorithm is the action of halving its congestion window in the face of a packet drop. This sudden decrease marks a discontinuity point on its evolution curve, and causes a fast and large increase in the RTT. As a result, the first property required for a smooth congestion control is that its curve should have either no discontinuity points or as few as possible. This requirement is called the *smooth curve property*.
- **Vertical Smooth Property:** Formally, the congestion control uses a function *f(u)* to translate events u into window sizes. When *f(u)* is a function of a variable where bursts with very large amplitude can occur, like RTT, this cannot be done linearly, since variations of RTT near 0 are more important or significant than variations among large values. For this reason, a hyperbolic tangent function is proposed to introduce vertical smoothness, making sure certain signals remain within a specified range.
- **Horizontal Smooth Property:** Sometimes it is important to know the frequency of an event (or its rate) instead of its amplitude for congestion control. Consequently in *f(u)*, u should mean the rate of those events. In our work u is taken to be the smooth average rate as described in [12][1].
- **Proactive Property:** Suppose a packet is dropped at a router because its queue is full. In order for the sender to decrease its sending rate, the receiver sends duplicate acknowledgment packets which arrives approximately 1 RTT (more precisely, 1 RTT and 3 duplicate acknowledgment arrivals) in the sender. This relatively long time, which we refer to as a ***temporal gap***, is one of the principal reasons TCP reacts by halving the window [7].

  We can do better if other metrics are used to signal congestion. Our work proposes the use of the following metrics: RTT (details the relation between the router queue size and RTT can be found in [13]) and ICMP-Source Quench (ICMP-SQ) Messages. Explicit Congestion Notification (ECN) [14] can be used instead of ICMP-SQ, but the latter has a smaller temporal gap since the messages are to be sent directly to the sender (in the opposite direction).
- **Precision Property**: Another problem is that a packet drop is not always the best way to signal congestion. A packet drop does not always mean that the packet was discarded as a result of congestion in a router queue. For example, a packet could be dropped because of a CRC error. For that reason, we say that there is a *signal uncertainty*. Algorithms like Eifel [3] and F-RTO

[1] The smooth average of n terms is:
$\bar{u}(u_n)=(1-\alpha)*\ \bar{u}(u_{n-1})+\alpha*u_n$ where α is 1/8 or ¼.

[6] eliminate some of the uncertainties, but not completely. The possibility of using ICMP-SQ messages or ECN in a proactive manner also enhances the precision of the congestion signaling. A router only sends them when the threshold of a RED queue is reached. On the other hand, RTT increases could be provoked by events other than congestion, like a router table update. Therefore, RTT would not be as precise as ICMP-SQ or ECN messages.

B. SmoothTCP Algorithms

For our notion of smoothness, we would like to be able to make infinitesimal adjustments *dw* to the current congestion control window w_t, that is, the new window w_{t+dt} after the infinitesimal time *dt* would be given by:

$$w_{t+dt} = w_t + dw \tag{1}$$

For that, we think of the adjustment *dw* as being defined as a function of some set of variables $p_1,..,p_i,..,p_n$ describing the network state and congestion conditions. Let us temporarily define *dw* = *D*. Furthermore, let us use only *w* to denote w_t since most of our functions do not use time (*t*) as independent variable. Specifically, we are interested in changes in *w*, that is an increment or decrement to the existing window size depending on changes in $p_1,..,p_i,..,p_n$.where *D* is formulated as:

$$D = \frac{\partial w}{\partial p_1} dp_1 + ... + \frac{\partial w}{\partial p_i} dp_i + ... + \frac{\partial w}{\partial p_n} dp_n \tag{2}$$

As per our previous discussion of "smoothness", we assume that each partial derivative is of the form:

$$\frac{\partial w}{\partial p_i} = A_{p_i} + C_{p_i} \tanh(B_{p_i}(p_i + M_{p_i})) \tag{3}$$

Equation (2) defines a family of functions. When such a family includes only smooth functions, we call it a *Family of Smooth Congestion Control Algorithms* (or briefly, *Smooth Algorithms*).

B. Fairness

In order to enforce SmoothTCP connections sharing approximately the same bandwidth, the following idea, inspired from Standard TCP, is also used to ensure fairness. If *D* is the increment defined in (2), then the final value *dw* added to w in (1) is redefined as follows:

$$\begin{aligned} D > 0 &\Rightarrow dw = D/w \\ D \le 0 &\Rightarrow dw = D \times w \end{aligned} \tag{3}$$

Consequently, having n connections sharing different window sizes, after some finite number of changes in $p_1,..,p_i,..,p_n$, the n connections should have approximately the same window size. This is because the connections having the largest windows receive the smallest window increments. In contrast, the connections having the smallest windows sizes suffer the smallest decrements.

IV. SMOOTHTCP

Basically, SmoothTCP is different from Standard TCP in its slow-start and congestion-avoidance algorithms. Indeed, it was designed to provide a smoother window size variation than Standard TCP.

A. Congestion Avoidance

We can then introduce the form of the functions for the partial derivatives that define the particular instance of Smooth Algorithms called *SmoothTCP*:

$$D = \frac{\partial w}{\partial j} dj + \frac{\partial w}{\partial x} dx + \frac{\partial w}{\partial f} df + \frac{\partial w}{\partial e} de + \frac{\partial w}{\partial q} dq \tag{4}$$

Where *j* is the variation in RTT, *x* is the variation in the number of timeout retransmits, *f* is the variation in the number of fast retransmits, *e* is the variation in the number of ECN acknowledgments packets and q is the variation in the number of ICMP-SQ messages. Then, the individual partial derivatives are:

$$\begin{aligned} \frac{\partial w}{\partial j} &= A_j + C_j \tanh(B_j(j + M_j)) \\ \frac{\partial w}{\partial x} &= A_x + C_x \tanh(B_x(x + M_x)) \\ \frac{\partial w}{\partial f} &= A_f + C_f \tanh(B_f(f + M_f)) \\ \frac{\partial w}{\partial e} &= A_e + C_e \tanh(B_e(e + M_e)) \\ \frac{\partial w}{\partial q} &= A_q + C_q \tanh(B_q(q + M_q)) \end{aligned} \tag{5}$$

Therefore, SmoothTCP is defined as a family of functions where any instance can be a function of these five metrics. When the corresponding partial derivative for a certain metric in (4) or (5) is different from 0, that metric is called a ***control metric***.

A. The Stable State

When the network does not suffer any alteration of state, such as the addition or deletion of TCP connections or other types of flows, each SmoothTCP connection converges to a particular window size. This happens because all variables involved in (4) do not undergo any changes, that is, *dj=dx=df=de=dq*=0. So the corresponding window increment dw is zero, dw=0.

Even if all SmoothTCP connections reach the convergence point, they can end up having different window sizes. These values for the window sizes will not be modified unless some event provokes changes in the network bandwidth. Having different converged window sizes results in different bandwidths for each SmoothTCP connections.

B. Fairness Factor

A new mechanism should be provided allowing SmoothTCP to get out of the stable state and continue its search for a shared bandwidth which is approximately the same for all SmoothTCP connections having the same

origin and destination. Adding a new increment κ to D that is not dependent on any increment of the variables in (4) does the trick:

$$D = \frac{\partial w}{\partial j}dj + \frac{\partial w}{\partial x}dx + \frac{\partial w}{\partial f}df + \frac{\partial w}{\partial e}de + \frac{\partial w}{\partial q}dq + \kappa \quad (6)$$

κ is called ***fairness-factor***. Its effect is to add small increments to *dw=D/w* (see (3)) and thus to w in order to increase the sending rate of the connection each time there is a new acknowledgment packet. As the receiving rate of the connection is kept constant, the queue of the bottleneck router will start to retain some packets and eventually become overloaded. As a result, congestion signals would be sent and the sender would start to decrease w by $dw=D \times w$ (see (3)). The constant κ can be thought as a function of the number of acknowledgment arrivals.

C. Slow-Start

The congestion avoidance of SmoothTCP starts after a certain window size is reached. Therefore, SmoothTCP needs to have a slow-start procedure like the one in TCP. Furthermore, the slow-start procedure of SmoothTCP needs to converge to a certain point in order for its congestion avoidance to take control of the window size. To do this, we also use a hyperbolic tangent (*tanh*) to implement our new slow-start procedure. Using the number of packet acknowledgments (*a*) arriving at the sender as independent variable, and $A_a = -C_a$ in (3) becomes:

$$dw = A_a - A_a \tanh(B_a(a - M_a)) \quad (7)$$

Considering the initial value for the number of packet acknowledgments as 0 and the initial window size as 1 MSS, (7) can be transformed to:

$$w = \sqrt{2[a.A_a - \frac{A_a}{B_a}\ln(\cosh(a.B_a)) + T]} - \sqrt{2T} + 1 \quad (8)$$

The function given by (8) converges to a fixed point given by the limit:

$$w_c = \lim_{a \to \infty} w = \sqrt{2\left(\frac{A_a}{B_a}\ln(2) + T\right)} - \sqrt{T} + 1 \quad (9)$$

Consequently, the new slow-start algorithm can be defined as:

```
slow_start=true
WHILE slow_start AND newAck() DO
    IF(w<Ws) AND (no_congest_signal)THEN
        IF(congestion_signal)THEN
                d = w_c – w
                w_c = lim_{a→∞} (w)   //(Eq.9)
                A_a= A'_a (d)         //(Eq.10)
                w = w/2
                a = W^{-1}(w)
        ELSE
                w = w+A_a-C_a tanh(B_a(a+M_a))
                a = a+1
    ELSE
                slow_start = false
END WHILE
```

The function *newAck()* waits for the arrival of a new packet acknowledgment. When any acknowledgment arrives, the algorithm continues to the next step. The objective of the new slow-start algorithm is to match the value of this limit to a stable window. Initially, let W_c be the limit given by (9) for some initial A_a. Each of the iterations tests a new function. That function has the same coefficients B_a and M_a, but different coefficients A_a. In each of the iterations (see Fig. 1), the window size is continuously incremented by *dw* (see (7)). When any congestion signal (packet drop) is detected, a new value for A_a, A'_a, is calculated which starts a new iteration. If the window size reaches W_c and no congestion signal is detected, then the slow-start function has found the stable window and the procedure terminates.

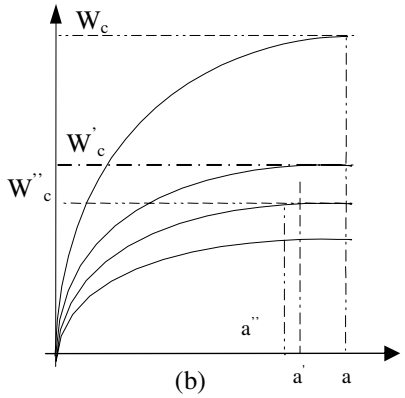

Figure 1. Slow-start evolution.

For computing the new value for A'_a for A_a in the slow-start algorithm, we used the function:

$$A'_a = \frac{A_a \ln(2) - B_a d\sqrt{2\left(\frac{A_a}{B_a}\ln(2) + T\right)} + \frac{B_a d^2}{2}}{\ln(2)} \quad (10)$$

Where *d* is the difference between the current value for w_c and the current window *w*. Finally, for calculating the new value for *a* corresponding to a given value *w/2* for the window w, the inverse function of w is used:

$$a = \frac{\ln\left(e^{\frac{-B_a}{2A_a}\left[\left(w + \sqrt{T} - 1\right)^2 - 2T\right] + \ln(2)} - 1\right)}{-2B_a} \quad (11)$$

Since it is out of the scope of this work, we are not going into more details about the slow-start procedure. However, the work in [10] contains more details about the slow-start procedure, including some discussion about its performance.

V. NORMAL RTT BEHAVIOR OF SMOOTHTCP

Using these equations and algorithms, we have simulated, using Network Simulator (NS-2) [15] diverse scenarios of the model of a network with a bottleneck link as shown in Fig. 2. The simulation time used was 600s.

All these simulations have the sender transmitting constant-size packets of 586 bytes and the SmoothTCP coefficients used were A_a=*40*, B_a=*0.003038*, A_q=*0.0*, B_q=*0.5493*, C_q=*0.0625* and M_q=*0.0*. All SmoothTCP connections have fairness-factor of κ=0.005. The link between the router and the sender is 1 Mbps with latency of 1ms and between the router and the receiver is 0.03125Mbps with latency of 8ms. Modified drop-tail queues of 20-packet size were implemented in the router in order to send ICMP-SQ messages when the queue size reaches the threshold (τ) of 7 packets. That means that, when the size of the queue is below τ, no ICMP-SQ messages are sent. However, an ICMP-SQ message is sent for each packet arriving in the router queue if the size is equal or above.

On the graphics showing the simulations result presented next, we have the following legend:

- **Congestion window**: "cwnd" is used to note the congestion window for Standard TCP and SmoothTCP. Additionally, the number of the connection (1 or 2) is concatenated to both strings. Both values are given in packets.
- **RTT**: RTT values are given in seconds and noted as RTT in the graphics for both Standard and SmoothTCP.
- **Current queue size**: The queue size in the router is given in packets and the string "size1" is used to note it in the graphics.
- **Number of ICMP-SQ messages:** The string "nicmpsq"is used to note the number of ICMP-SQ messages that arrives in both senders (Standard TCP or SmoothTCP). Additionally, the number of the connection (1 or 2) is concatenated to the strings.
- **Packet Drops:** The number of packets dropped in the router is noted by "pdrops1" independently on the type of the connection (Standard TCP or SmoothTCP).
- **Packet Arrival Rate:** The number of packets arriving in the router per second, it is noted by the string "parvlr1".

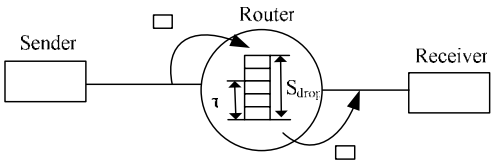

Figure 2. Bottleneck link model.

A. Standard TCP

Fig. 3, Fig. 4 and Fig. 5 illustrate the behavior of Standard TCP. The main results can be enumerated as follows:

1. **Congestion window and RTT**: In these experiments, the window size of Standard TCP and RTT varies a lot. Fig. 3 shows that the window size is periodically decreased (mostly halved) in both connections, indicating that the sender has detected a packet drop. Because of this behavior, the RTT also has huge variations, most of the time between 1.4 s and 3 s.
2. **Router queue size and the number of ICMP-SQ messages**: The router queue size also follows the behavior of the window size (see Fig. 4). Periodically, the queue is filled up and packets are dropped, after the queue size reaches a minimum. Both points happened after the window size in the sender reaches the maximum and the minimum. Regarding the number of ICMP-SQ messages sent by the router to the senders, the two curves are almost linear. That means the average of the queue size was above of the queue threshold.
3. **Packet drops and arrival rate of packets in the router queue**: The number of packets dropped by the router also follows an almost linear growth (See Fig. 5). For the arrival rate of packets, the spaces can be seen where no packets arrive since the sender is waiting for the packet dropped being confirmed.

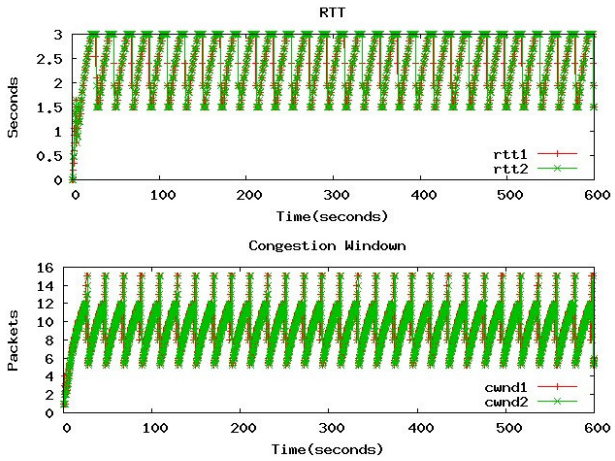

Figure 3 . RTT and congestion window in Standard TCP.

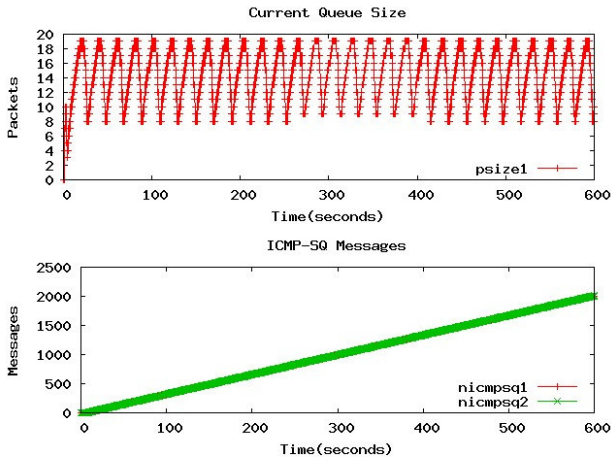

Figure 4. Router queue size and number of ICMP-SQ messages in Standard TCP.

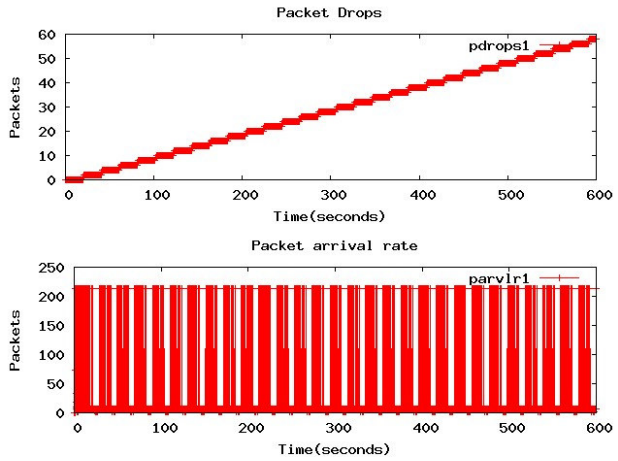

Figure 5. Packet drops and arrival rate in Standard TCP.

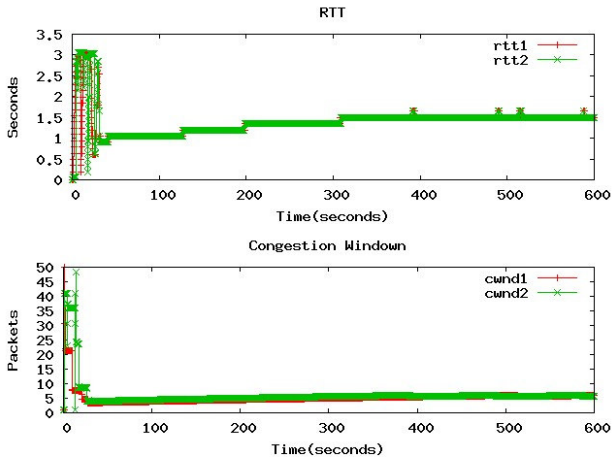

Figure 6. RTT and congestion window in SmoothTCP.

B. SmoothTCP

Fig. 6, Fig. 7 and Fig. 8 illustrate the behavior of SmoothTCP. Briefly, we have:

1. **Congestion window and RTT**: The window size of SmoothTCP and its RTT is smaller and has less frequent variations that Standard TCP (see Fig. 7). This characteristic of SmoothTCP is controlled by the fairness-factor (κ). Increasing the fairness factor also increases the variations in RTT.
2. **Router queue size and the number of ICMP-SQ messages:** SmoothTCP generally results in a smaller queue size than Standard TCP and the number of ICMP-SQ messages is almost the same as in Standard TCP (see Fig. 7). Because of the value of the queue threshold (τ), a large number of ICMP-SQ messages is sent by the router when the queue occupancy is above τ. If τ was chosen larger, a smaller number of ICMP-SQ messages would be sent, but the slow-start procedure would take more time to converge.
3. **Packet drops and arrival rate of packets in the router queue:** The number of packets dropped by the router when SmoothTCP is used is only large during the slow-start phase (see Fig. 8). No packets are dropped in the congestion avoidance phase. Having no packet drops in the congestion phase also contributes to ensuring small variations in RTT. In particular, there is a notable difference between the behavior of the packet arrival rates of SmoothTCP and Standard TCP. While in Standard TCP, large bursts of packet arrivals happen periodically, in SmoothTCP they are smaller and less frequent. Regardless, they have almost the same throughput. For Standard TCP, the first connection transferred 988,384 bytes and the second 1,192,600 bytes. For SmoothTCP, the first connection transferred 1,046,272 bytes and the second 1,230,656 bytes.

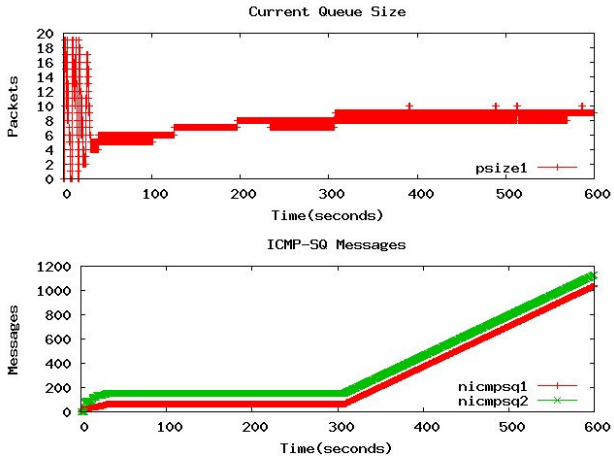

Figure 7. Router queue size and number of ICMP-SQ messages in SmoothTCP.

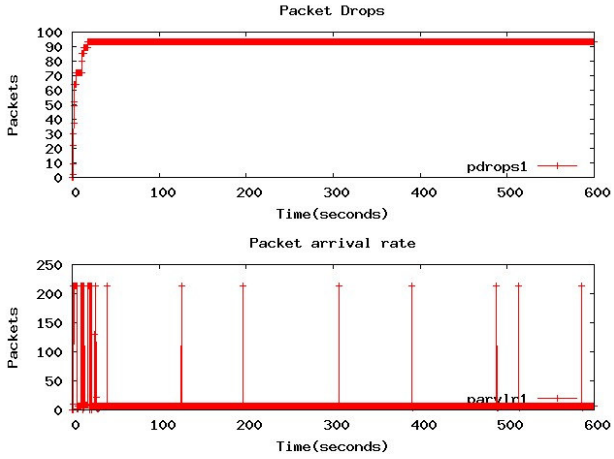

Figure 8. Packet drops and arrival rate in SmoothTCP.

VI. RTT Behavior of SmoothTCP in Connections Subject to Spurious Errors

The same bottleneck link shown in Figure 2 was simulated as having 3 spurious errors occurring on each connection during the 600s of the simulation. The time

where those errors happened was chosen randomly, although we enforced a large interval between them to isolate any effect that one can have in another. For these simulations, they happened at 81.13s, 254.14s and 427.19s.

A. Standard TCP

Fig. 9 and Fig. 10 illustrate the behavior of Standard TCP. The main results can be enumerated as:

1. **Congestion window and RTT**: For these experiments, the window size of Standard TCP and the RTT varies significantly. Fig. 9 shows points where the window size is halved in both connections, indicating that the sender has detected a packet drop. Because of this behavior, we cannot clearly identify where the spurious errors happen or verify their effects. That means that Standard TCP masks the effects of packet drops and the spurious errors.
2. **Router queue size and the number of ICMP-SQ messages**: In these experiments, the router queue size also follows the behavior of the window size as shown by Fig. 10. The number of ICMP-SQ messages sent by the router to the senders are nearly linear over the time of the simulation, i.e., the two curves in Fig. 10 are almost linear. Likewise, it is not clear where are the effects of spurious errors on the queue size and the number of ICMP-SQ messages.

B. SmoothTCP

Fig. 11 and Fig. 12 illustrate the behavior of SmoothTCP. Briefly, we have:

1. **Congestion window and RTT**: For these experiments, the window size and RTT of SmoothTCP are smaller and have less frequent variations than Standard TCP. Fig. 11 shows three points where there are large decrements in RTT. Immediately before each of those points, there is a spurious packet error and one packet is lost. When the lost packet is retransmitted, all the acknowledgments, for packets sent after it, are confirmed at once. It causes the RTT decrement. In the Appendix A there are more results showing the effect in RTT variation for other values of κ.
2. **Router queue size and the number of ICMP-SQ messages:** When a spurious error occurs, the router queue size suffers a large decay in its queue size. This is because the sender does not send any further packet once the number of them sent, and not confirmed, has reached the window size. Therefore, the sender keeps waiting for the acknowledgment of the retransmitted packet what makes the occupancy of router queue be reduced.

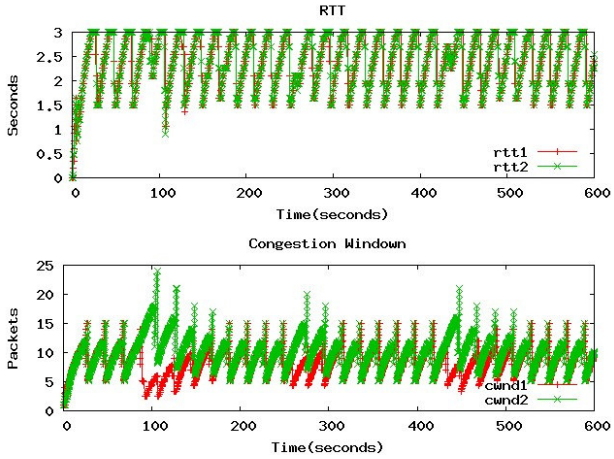

Figure 9. RTT and congestion window in Standard TCP.

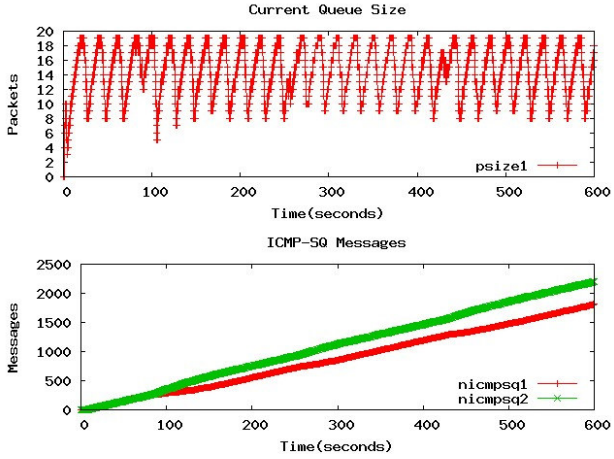

Figure 10. Router queue size and number of ICMP-SQ messages in Standard TCP.

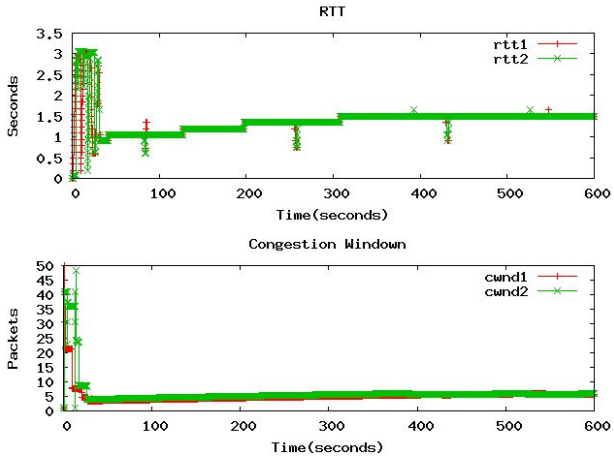

Figure 11. RTT and congestion window in SmoothTCP.

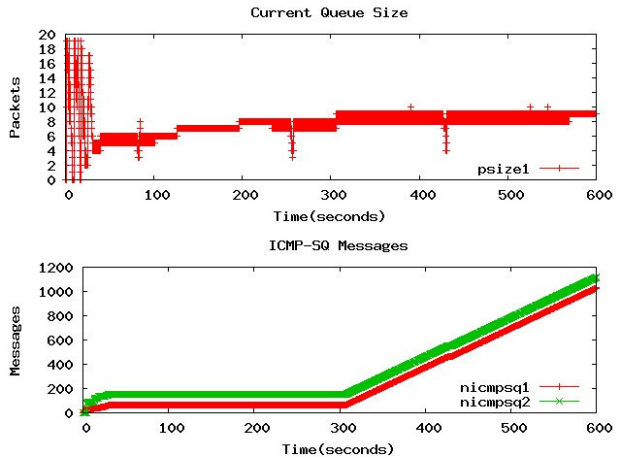

Figure 12. Router queue size and number of ICMP-SQ messages in SmoothTCP.

VI. Final Considerations and Conclusion

Showing what specifically happens when the parameters of SmoothTCP are changed is beyond the scope of this paper, but it can be seen in [9,10,11]. However, we can briefly summarize those effects as:

- **Fairness-factor (κ):** This parameter together with coefficient C_q is responsible for the adjustment of the fairness characteristic of SmoothTCP. Although other parameters have some effect on RTT variation, κ seems to have the greatest influence. By increasing κ, we also increase the RTT variation. In our simulations, we concluded that SmoothTCP connections with κ values from 0.001 until 0.005 result in RTT variation much smaller than TCP.
- **B_a:** This parameter controls the convergence time of the slow-start procedure, that is, the duration of the slow-start. As the region of the congestion window curve in which the slow-start procedure acts is very unstable in terms of RTT variation, we might wish to have a short time for the slow-start.
- **Queue Threshold (τ) and Queue Size (S_{drop}) :** The difference (S_{drop}-τ) should be enough to accommodate the burst of packets following a retransmitted packet. If it is too small and packets are dropped as a consequence, RTT is also increased.

Therefore, in a network subjected to spurious errors we can adjust the RTT variation by basically adjusting κ. However, too small values of κ can lead to unfair bandwidth sharing by the SmoothTCP connections [10]. In our experiments, we have used values of κ between 0.001 and 0.005 with fairness comparable to Standard TCP and RTT variation as those illustrated in this paper.

Appendix A Fairness-factor Effect

The following figures show additional results of simulations performed in the environment shown by Fig. 2. Fig. 13 and Fig. 14 show the behavior of SmoothTCP when κ=0.005. Fig. 15 and Fig. 16 show the behavior when κ=0.010. In both set of simulations 3 spurious errors are inserted randomly. For this set of simulations, they happened at 81.13s, 254.14s and 427.19s, the same periods of time in the simulations presented before. All other configuration parameters are kept the same used in the previous simulations.

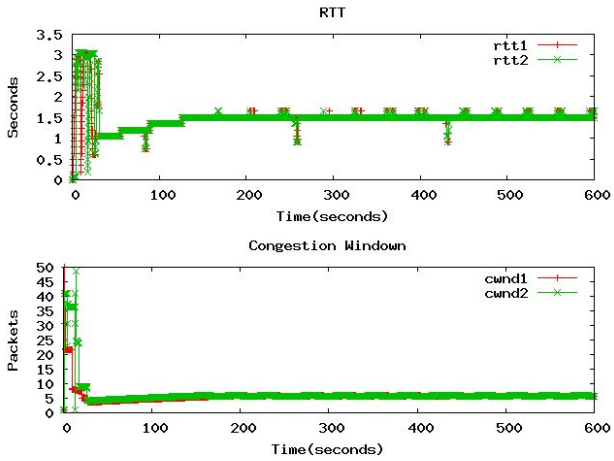

Figure 13. RTT Variation and congestion window of SmoothTCP when κ=0.005.

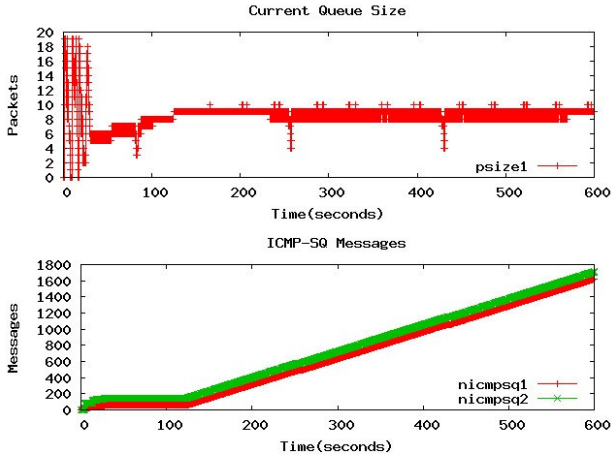

Figure 14. Router queue size and number of ICMP-SQ messages in SmoothTCP when κ=0.005.

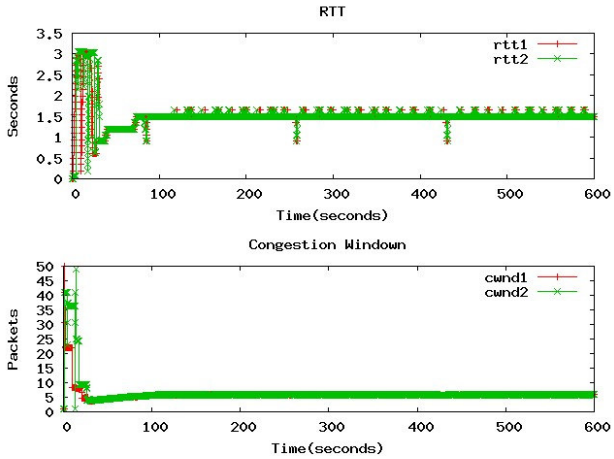

Figure 15. RTT Variation and congestion window of SmoothTCP when κ=0.010.

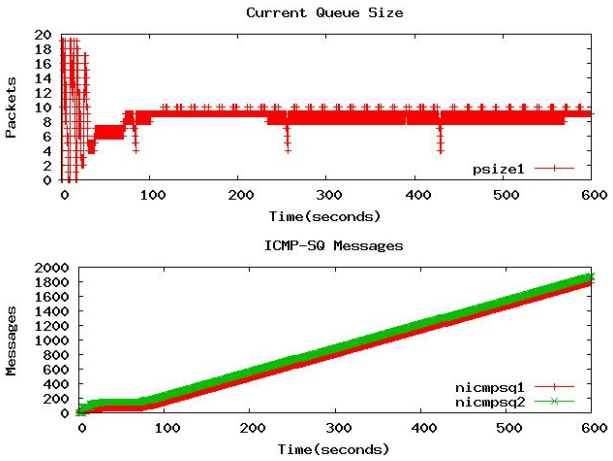

Figure 16. Router queue size and number of ICMP-SQ messages in SmoothTCP when κ=0.010.

REFERENCES

[1] J. Nagle. RFC0896: "Congestion Control in IP/TCP Internetworks", Network Working Group, IETF, Jan., 1984.

[2] S. Floyd, M. Handley, J. Padhye and J. Widmer. "Equation-Based Congestion Control for Unicast Applications: the Extended Version", Technical Report, TR-00-003, ICSI, Berkeley, CA, Mar., 2000.

[3] R. Ludwig and R. Katz. "The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions", *ACM SIGCOMM 00 Computer Communication Review*, Vol 30-1, Jan., 2000, pp. 30-36.

[4] M. Gerla, M. Sanadidi, R. Wang, A. Zanella, C. Casetti and S. Mascolo. "TCP Westwood: Congestion Window Control Using Bandwidth Estimation", *Proceedings of IEEE Globecom 2001*, Vol 3, San Antonio, Texas, Nov., 2001, pp. 1698-1702.

[5] Zanella, G. Procissi, M. Gerla, and M. Y. Sanadidi. "TCP Westwood: Analytic Model and Performance Evaluation", *Proceedings of IEEE Globecom 2001*, San Antonio, Texas, Nov, 2001, pp. 1703-1707.

[6] P. Sarolahti, M. Kojo and K. Raatikainen. "F-RTO: an enhanced recovery algorithm for TCP retransmission timeouts", *ACM SIGCOMM 03 Computer Communication Review*, Vol 33-2, ACM Press, 2003, pp. 51–63.

[7] V. Jacobson. "Congestion Avoidance and Control", *ACM SIGCOMM 88 Computer Communication Review*, ACM Press, 1988, pp. 314-328.

[8] D. Bansal and H. Balakrishnam. "TCP-Friendly Congestion Control for Real-time Streaming Applications", Technical Report, MIT-LCS-TR-806, M.I.T Laboratory for Computer Science, Cambridge, MA, May, 2000.

[9] E. Vieira and M. Bauer. "Smoothness Properties in Congestion Control for TCP", *3rd International Conference on Computing Communications and Control Technologies*, Austin, USA, Jul, 2005.

[10] E. Vieira and M. Bauer. "Smooth TCP", *Proceedings of the IV Latin American Network Operations and Management Symposium (LANOMS 2005),* Porto Alegre, Brazil, Aug, 2005.

[11] E. Vieira and M. Bauer. "Smooth TCP", *Proceedings of the IEEE Consumer Communications and Networking Conference (CCNC 2005)*, Las Vegas, USA, Jan, 2006.

[12] V. Paxson and M. Allman. RFC2988: "Computing TCP's Retransmission Timer", Network Working Group, IETF, Nov., 2000.

[13] G. Appenzeller, I. Keslassy and N. McKeown. "Sizing Router Buffers", *ACM SIGCOMM 04 Computer Communication Review*, 2004, pp. 281-291.

[14] K. Ramakrishnan, S. Floyd and D. Black. RFC3168: "The Addition of Explicit Congestion Notification (ECN) to IP", Network Working Group, IETF, Sep., 2001.

[15] K. Fall and K. Varadhan. "The ns Manual", http://www.isi.edu/nsnam.

ACKNOWLEDGMENT

This work was supported in part by a PhD scholarship from CNPQ, Brazil.

Elvis M. Vieira received the B.S degree in computer science from Federal University of Santa Catarina, Brazil, in 1989. He also received the M.S. degree in computer science computer from Federal University of Santa Catarina, Brazil, in 1996. Currently, he is a PhD candidate in computer science at The University of Western Ontario, Canada.

After his graduation, he has been working in the IT Department of Federal University of Santa Catarina, Brazil, as System Administrator. He also authored and co-authored several papers in application network management and, lately, in congestion control of TCP. His research interests cover the performance evolution, design and control of distributed computer communication systems, wireless and scheduling algorithms.

Michael Bauer received his PhD from University of Toronto, Canada.

He is a Professor of and Chair of the Department of Computer Science at the University of Western Ontario. He was also Chair of the Department from 1991-1996. From 1996-2001 he was the Associate Vice-President Information Technology at the University of Western Ontario. He has published over 100 papers on various topics in the computing field and has collaborated with a number of companies on joint research projects. His research interests include distributed computing, particularly the management of distributed applications and systems, network management, software engineering, and high performance computer networks.

Dr. Bauer is a member of the IEEE and the Association for Computing Machinery (ACM) and has served on various committees of both organizations. He has also served on the organizing and program committee of numerous conferences and has refereed for a variety of international journals.