Enhancing TCP Performance in Networks with Small Buffers

Ashu Razdan, Alok Nandan, Ren Wang, Medy Sanadidi and Mario Gerla
Network Research Laboratory
Computer Science Department
University of California, Los Angeles
Email:{razdan,alok,renwang,medy,gerla}@cs.ucla.edu

Abstract— **TCP performance can be severely affected when the buffer capacity is small. This is possible either because many flows share the network or that the bandwidth-delay product is large (e.g. satellite links). The behavior of various versions of TCP with respect to buffer capacity issues has not been studied in much detail. In this paper, we investigate the behavior and performance of different TCP variants under small buffer capacity conditions. We recognize TCP pacing as a potential solution. However, instead of using TCP's sending rate as the dictating metric, we make use of the bandwidth-share estimate (BSE) maintained by TCP Westwood, to set the pacing interval. We call this newly proposed protocol *Paced-Westwood*. We also show the need to scale BSE further to mitigate the effects of positive feedback in BSE. For this, we propose a further enhancement that we call α-paced Westwood that uses a scaling parameter α to enforce convergence of BSE and the pacing interval.The proposed α-paced Westwood uses its BSE to space the packet bursts during the slow-start phase, resulting in a superior throughput in the troublesome low buffer capacity cases. With the help of simulations we show that our enhanced TCP Westwood outperforms a both unpaced as well as paced TCP NewReno under low buffer capacity networks.**

Keywords— **TCP Performance, small buffers**

I. Motivation

TCP is the most widely deployed transport layer protocol in the Internet today. TCP was originally designed to run over a variety of communication links. However, recent advances in satellites and fibre-optic networks have caused us to re-evaluate TCP's purported flexibility. One problem that is common to both satellite and fibre-optic networks is that the capacity of these networks can be much greater than in conventional networks. The mismatch between the high capacity of these networks, and available storage at the queues of individual network routers poses problems for TCP. Given the ever-growing interest in these new network technologies and the centrality of TCP in network communication, the investigation done in this work was imperative.

A. The Problem

TCP uses two algorithms, namely *Slow Start (SS)* and *Congestion Avoidance (CA)*, to control the flow of packets in the Internet [8]. With SS, the Congestion Window (*cwnd*) is set to one segment and is incremented by one segment for every non-duplicate ACK received. If we suppose the window advertised (*awnd*) by the receiver does not limit the source throughput, this process continues until the SS threshold (W_{th}) is reached or losses occur. If W_{th} is reached without encountering any losses, the source moves into a more conservative approach of congestion avoidance where it increases the Window (W) by one packets for every window of ACKs. Otherwise, W and W_{th} are both reduced and a recovery phase is called. TCP supposes this new W_{th} to be a more accurate estimate of the network capacity.

In this paper we present an investigation of a problem that occurs when TCP operates in a network having a small buffering capacity compared to its bandwidth-delay product. It is the problem of loosing packets during SS before fully utilizing the available bandwidth. To understand this particular problem, consider the following idealized behavior of TCP during SS. During SS, for every segment ACKed, the sender transmits two new segments. In effect, this behavior means the sender is transmitting at *twice* the data rate of the segments being ACKed. And keep in mind the separation between ACKs represents (in an ideal world) the rate segments can flow through the bottleneck router in the path. So the sender is bursting data at twice the bottleneck rate, and a queue must be forming during the burst. In the simplest case, the queue is entirely at the bottleneck router, and at the end of the burst, the queue is storing half the data in the burst. [1]

If the network buffers are not designed to absorb this high rate, they will overflow and losses are detected before the source gets into congestion avoidance. The window size when these losses are detected is a wrong estimation of the network capacity. But TCP considers it as the maximum reachable window which results in a throughput degradation. This problem, known as the double SS phenomenon, has been studied previously in [9]. It can be seen in any SS phase if the buffer capacity does not scale with the bandwidth-delay product of the network. Also it can be seen in the first SS of the connection and if the default values are bad then the performance of short-lived connections are affected. A phenomenon of three consecutive SS phases per cycle has also been observed and analyzed in [3]. Due

[1] During the burst, we transmitted at twice the bottleneck rate. Suppose it takes one time unit to send a segment on the bottleneck link. During the burst the bottleneck will receive two segments in every time unit, but only be able to transmit one segment. The result is a net of one new segment queued every time unit, for the life of the burst.

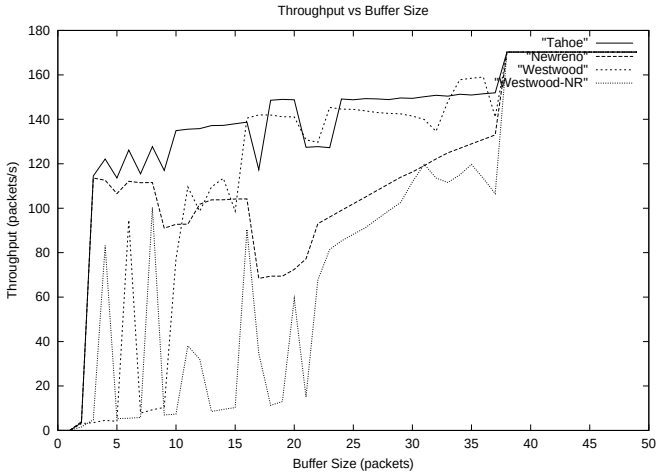

Fig. 1. *Throughput Vs Buffer Size for various TCP variants*

to the multiplicative decrease of the congestion window after a loss detection, these consecutive slow-starts results in a deterioration in TCP throughput. This phenomenon occurs in a cyclic manner in networks having a small buffering capacity compared to their bandwidth delay product. This phenomenon [3] has been observed in TCP Tahoe, as seen in the graph 1.

The analysis of the double and triple slow-start phenomenon using the fluid flow approach has been done in [9] and [3] respectively which gives us the necessary conditions for double (1) and triple slow-start (2) below where B is the buffer size and μT is the pipe size.

$$\frac{B}{\mu T + 1} < \frac{1}{3} \tag{1}$$

Intuitively too, the triple slow-start phenomenon would occur when the buffers are much more constrained compared to the case of double slow-start as is confirmed by the two analytic results.

$$\frac{B}{\mu T + 1} < \frac{1}{7} \tag{2}$$

Also note from the graph 1 the value at which the triple slow-start phenomenon starts degrading the throughput. Given the fact that the pipe size *i.e* μT is 107 packets. We observe that throughput degrades at the 15-20 packets mark that's approximately the one-seventh of 107. In Figure 1, we plot the throughput achieved by Tahoe, Newreno and Westwood in the case of small buffers. The curve corresponding to Tahoe conforms to the analysis done by Barakat etal. We notice that the dips are most prominent around buffer size equal to 15. Barakat etal. explain these dips by exposing the double and triple slow start phenomena occurring at buffer sizes equal to μT/7 (which is roughly 15 packets). However, similar inconsistencies in throughput can also be seen for Newreno and Westwood. This behavior warrants further analytical study for these TCP variants.

In the course of our investigation we discovered a mechanism that was initially proposed to improve performance in such a scenario. We propose our enhancement to a recent TCP variants which gels very well with this original idea proposed in [11].

The rest of the paper is organized as follows-Section II describes previous work in this area, specifically with reference to pacing [11]. Section III discusses briefly about one of the TCP variants we use before delving deep into our proposed enhancement.Section IV describes the experimental algorithms, simulation methodology and section V describes the simulation results. Finally, we conclude in section VI.

II. Related Work

Several solutions have been proposed to the above mentioned problem . One proposition [4] is to reduce the SS threshold W_{th} so that to get in CA before the overflow of buffers.The idea is to decrease the W_{th} to bring it below the overflow window W_B [4], [3] . This makes the source enter CA with one SS without buffer overflow. Because W_{max} is imposed by the network parameters, the only possible way is to change the reduction factor one half used by TCP in the calculation of W_{th} . However this scheme does not always work, in certain cases where the ratio on the left side of the equation (1) falls below 1/4, it does not work as proved in [4]. However, another more interesting and powerful idea [4] is *packet-spacing*. The idea is to keep W_{th} unchanged and to increase W_B so that to get in CA before buffer overflow. We know W_B is inversely proportional to the queue building rate. To increase it, the packets that TCP sends in bursts during SS is *spaced*.

A. Is Pacing the Solution?

This idea of *packet spacing* is not new. Earlier work of [7] to support higher throughput for Persistent-HTTP connections over TCP uses the idea of rate-based pacing (RBP). Partridge argues [11], [1] that pacing can address problems in TCP performance on long-latency, high bandwidth satellite links while the Berkeley WebTP group has combined pacing, receiver -driven congestion control, and application-level framing into a transport protocol specialized for web traffic [12]. However all the above implementations of Pacing to counteract the buffer overflow problem lack in one regard. What should be the rate at which the packets should be paced? Some of the implementations estimate it from the incoming ACK rate, while others use the latest RTT values to predict the optimum pacing parameter.

A.1 TCP Pacing

Pacing is a hybrid between pure rate control and TCP's use of ACKs to trigger new data to be sent into the network. As a traditional window based protocol, TCP uses a window to determine the number of packets that can be sent and uses the receipt of acknowledgments to trigger the sending of packets. Pure rate based schemes, on the other hand, use rates to determine both *how much* and *when* to send. Pacing is a hybrid between these two approaches, it uses the TCP window to determine *how much* to send but

uses rates instead of acknowledgments to determine *when* to send [11]. Instead of sending an entire window of packets in a single burst at the beginning of each round-trip time, the TCP sender should send out the packets in a steady stream over the entire course of a round-trip time.[2] In fact, under the ideal conditions, a Paced TCP implementation will only cause queuing bottlenecks to occur when the TCP sender is genuinely sending at a rate that is too fast for the server itself. Pacing is an attractive solution to the queuing bottleneck problems for at least two reasons. First, it relieves network designers from having to guess at buffer sizes based on typical bandwidth-delay products. Second, it can be implemented by modifying TCP senders only. Pacing does not require participation of the network's intermediate routers or TCP receivers.

A.2 TCP Paced-Newreno

For TCP Newreno, the pacing interval is dynamically set to the TCP sender's instantaneous rate [11]. Due to the fact that TCP Newreno does not have a bandwidth estimation mechanism, the only way the pacing interval can be set is through a very basic computation.

$$\delta = \frac{1}{\frac{CWND}{latestRTT}} \tag{3}$$

It should be noted that the pacing rate is being set as the rate of the TCP sender irrespective of the bottleneck bandwidth!

This observation suggests using Pacing along with a instantaneous sender rate estimator as a potential solution. In the next section we propose our enhancement leveraging this idea of instantaneous rate estimation using one of the newer TCP variants.

III. Our Proposed Enhancement

Before we get into our proposition we give a brief background on TCP Westwood, one of the newer TCP variants that uses the idea of rate estimation to set the congestion window.

A. Background

TCP Westwood (TCPW in short) [6], [13] is a sender-side only modification of the TCP Reno protocol stack that optimizes the performance of TCP congestion control over both wireline and wireless networks. TCPW is based on end-to-end bandwidth estimation to set congestion window and slow start threshold after a congestion episode, that is, after three duplicate acknowledgments or a timeout. The bandwidth is estimated by properly low-pass filtering the rate of returning acknowledgment packets. The rationale of this strategy is simple: in contrast with TCP Reno, which halves the congestion window after three duplicate ACKs, TCP Westwood sets a slow start threshold and a congestion window which tries to be consistent with the bandwidth used at the time congestion is experienced. Resetting the window to match available bandwidth makes TCPW more robust to sporadic losses due to transmission channel problems (e.g. satellite links, wireless links). These often cause conventional TCP to over-react, leading to unnecessary window reduction.

B. Some Observations

Intuitively, TCP Westwood and Pacing should work together and *complement* each other because of two reasons-

- Sender-side modifications: Both the Pacing mechanism and Westwood are sender side [5] modifications [3]. So the implementation and deployment of the two mechanism side-by-side is no problem at all.
- Feedback Effect: Pacing uses Westwood bandwidth estimation and Westwood uses the pacing mechanism to better estimate the bandwidth measurement, so the two mechanism work $in-tandem$ and help enhance the performance of each other.

C. Our Proposal

Our proposed enhancement *augments* the pacing scheme by using the Bandwidth-Share-estimate (BSE in short) of TCP Westwood to determine *how much* to send and *when* to send, see section II-A above. Most of Pacing implementations use the pacing mechanism across the lifetime of the connection. However our proposition is that we use Pacing only during the slow-start phase of the TCP window evolution.

TCP pacing is an elegant solution to the small buffer problem. Pacing can be implemented through the integration of a Leaky Bucket with a TCP sender. The token rate for the leaky bucket dictates the pacing interval. In TCP pacing, this packet inter-departure time is not a fixed parameter. The pacing interval changes dynamically based on the current sending rate of the TCP sender. For TCPW the pacing rate should be the same as the bandwidth share BSE measured from returning ACKs.

C.1 Initial Enhancement: TCP Paced-Westwood

In the previous subsection, we saw that the pacing interval for TCP Newreno was dependent on the TCP sender's instantaneous rate, irrespective of the bottleneck's bandwidth. However, Westwood maintains this measure in the BSE parameter. We expected that a reasonable pacing rate as indicated by this bandwidth estimates should work. For this, we use ABSE estimates instead of the BE proposed in the original TCPW.

However, there is a hazard in setting the pacing interval based purely on the BSE. This is because we expected to see some feedback effects, because if we over-estimate the

[2]For example, if the current TCP window is 8 packets and the RTT is 500ms, then the TCP sender should send out 1 packet every 62.5ms is enough time for a 3 Mb server to process a single packet, and therefore the queue in the network should never build up.

[3]By the definition however, pacing can be both a *sender* or a *receiver* side modification. Of course, receiver pacing is less effective, since acknowledgments arriving at the sender can trigger multiple data sends; with receiver pacing, these packets will be sent in a burst. Further, receiver pacing is susceptible to ACK compression.

pacing interval (TCPW starts off with a not-so-accurate BSE), it would affect the BSE (due to an increased ACK spacing), which in-turn would again affect the pacing interval (as we plug in the pathological BSE samples in the pacing scheme)! So we ought to be very careful, or else we might get stuck in a local maximum for the pacing interval. The simulation study in next section confirm our intuition.

C.2 Second Refinement: TCP α-paced Westwood

As pointed out in the previous subsection, the positive feedback effect was suspected to cause potential divergence in the BSE estimates. To mitigate this, we used the following scaled BSE to calculate the packet inter-departure times.

$$RE = \alpha BSE \tag{4}$$

It should be noted that having RE greater than BSE does not mean that the rate of TCP is RE, but it means that the bursts are being sent out at RE. These are clearly different. However, we do need a higher RE than BSE, or else the pacing intervals (and BSE as well) will not converge!

Now, if we set RE with alpha greater than 1.0, although we are not being conservative, we are pacing the connection! What we do achieve here is remove the instability of BSE due to the feedback effects. That we only use a "low-pass" filter in TCPW is one of the problems. If we are to remove the instability without using the constant alpha (larger than 1.0, which is not foolproof), we shall need to insert a "band-pass" filter instead. This is because it is the high pacing interval (high ACK spacing) that is killing us.

$$\delta = \frac{1}{RE} \tag{5}$$

We also compared the performance of α-paced Westwood with Paced-Newreno. These results are included in the next section.

D. Implementation

We have incorporated pacing into the `ns-2` [10] simulation code for all TCP variants. The `ns-2` code closely models the congestion control behavior of all of the TCP implementations in widespread use, including TCP Newreno as well as newer variants like TCP Westwood. We have enabled delayed acknowledgments for this study as well, unlike the pacing implementation in [2]. Our implementation of pacing uses a variant of the leaky bucket algorithm and is very similar to the implementations in [11]; one difference being that these implementations use pacing throughout the lifetime of a flow while we use it during specific periods (e.g. after idle time or at connection start up) ; another difference is that Timeouts are scheduled at regular intervals of duration `RTT/BSE` (instead of `RTT/Window`) [11] , [2]. A packet is transmitted from the window whenever the timer expires. This ensures that packet transmissions are spread across the whole duration of the RTT.

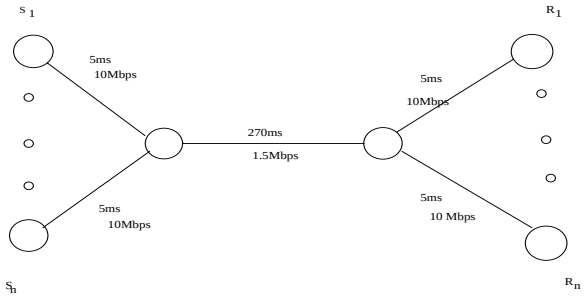

Fig. 2. *Simulation Setup for Multiple Flow study*

IV. Performance Evaluation

A. Experimental Algorithms

In order to study the effect of pacing on TCP Westwood, we examined two existing TCP algorithms with our proposed enhancement.

TCP NewReno is the most widely deployed TCP version in the Internet today. For TCP Newreno, the pacing interval is dynamically set to the TCP sender's instantaneous rate. Due to the fact that TCP Newreno does not have a bandwidth estimation mechanism, the only way the pacing interval can be set is through a very basic computation as shown in equation (3)

It should be noted that the pacing rate is being set as the rate of the TCP sender irrespective of the bottleneck bandwidth! As we see in the results, the bandwidth-share estimate (BSE) maintained by TCP Westwood proves invaluable to set the pacing rate.

B. Simulation Scenario

The simulation setup for the single flow is shown in Figure 3.The topology used for the multiple flows case is shown in Figure 2. One or more TCP connections are established between a set of senders and receivers through a single bottleneck link. The bottleneck link uses FIFO scheduling and drp tail buffer management.

We evaluate the impact of pacing on the aggregate throughput for all flows as well as individual flow throughputs. We also measure the fairness using a modified version of Jain's fairness index as suggested by [2] for flows with variable RTTs referred to as *Normalized Fairness*.

The scenario consists of three nodes connected in a linear topology. The source runs a TCP variant, with a persistent FTP connection at the application layer. The intermediate node acts as the bottleneck router. The source is connected to this bottleneck router with a high-speed link of rate 10Mbps. The bottleneck router is connected to the destination with a link of rate 1.5Mbps. The delays on these links are 10ms and 270 ms respectively, the latter being characteristic of a high-delay satellite network. These parameters give us a bandwidth delay product of 110Kb, or 107 packets. The metric chosen in this work for performance evaluation is TCP goodput. For the following analysis, however, we shall use the words goodput and throughput interchangeably. This metric is calculated as the total number of packets transmitted per unit time.

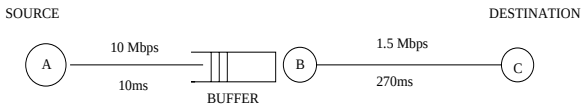

Fig. 3. *Simulation Setup for Single Flow study*

V. Results

The simulation results are organized as follows: section V-A proves the existence of the double and triple slow start phenomenon across all the TCP variants not just TCP Tahoe which [3] performed. Sections 8 and V-C presents results for single and multiple flows respectively.

A. Preliminary Simulations

In our first experiment, we worked with three well known TCP variants, namely Tahoe, Newreno and Westwood. We also worked with the latest version of TCP Westwood (Westwood-NR) that incorporate Newreno features like handling of multiple losses in a window. In this experiment, we plotted the throughput of these variants for a range of buffer sizes. We increase the buffer size from 0 to a maximum of 50, which is roughly one-half of the bandwidth-delay product. As can be seen in Figure 1, the throughput stabilizes and saturates around a buffer size of 38 packets. This value is well below one-half of the pipe size (107/2 packets/s). Thus, we observe that the throughput reaches it's maximum value at around one-half of the pipe size.

However, for very small buffer sizes, the throughput curve is seen to be highly unstable. In their analysis, Barakat etal used TCP Tahoe and exposed the double and triple slow-start phenomena that occur for buffer sizes around one-third the pipe-size. Similar patterns are also noticed for TCP Newreno and Westwood. For Newreno, we notice a drop at buffer size equal to 19, which is close to one-sixth of the pipe-size.

It would only be fair to illustrate to what extent pacing can help the small buffer scenario. Figure 4 compares the performance of a non-paced Newreno connection with a paced Newreno. Note that the buffer size chosen for this experiment was 2, which is extremely small. It can be seen in the plot that the non-paced connection goes into a series of slow-start phases. Eventually, the value of ssthresh for Newreno is set equal to 2. This is due to the fact that the connection experiences multiple drops for congestion window values greater than 2. However, the paced Newreno can pump away data comfortably, and reaches the congestion window value equal to 100, which is the maximum permissible value in our simulation.

B. Single Flow

We first evaluate the performance of Paced-Westwood with Paced-Newreno. Figure 5 illustrates the performance comparison of Paced-Newreno and Paced-Westwood. We expected Paced-Westwood to outperform Paced-Newreno. However, as it was noticed that for small buffer sizes the bandwidth estimation algorithm used by TCP-Westwood

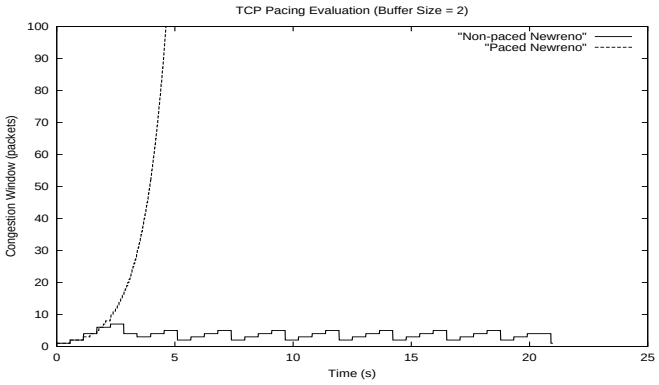

Fig. 4. *Pacing evaluation for TCP Newreno'*

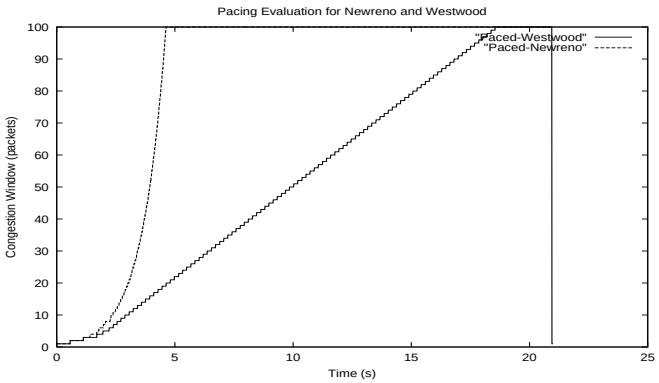

Fig. 5. *Pacing comparison for Newreno and Westwood*

does not perform well, the pacing interval value is set to a much larger value than it should be. This results in a overly underestimated BSE, and consequently, an overly overestimated pacing interval. Also, due to a positive feedback effect, the BSE algorithm always underestimates the bandwidth share, and this leads to the pathological case wherein Paced-Westwood sends out packets at a very low rate, due to the large pacing interval. This can be seen in Figure 5, where Paced-Newreno clearly outperforms Paced-Westwood, something contrary to what we though would happen. Our final refined α-paced Westwood mitigates the positive feedback effect of pacing by scaling the BSE estimated by Westwood by a factor α. This parameter α is typically larger than 1. To evaluate the performance of α-paced Westwood, we varied the values of α, and plotted its throughput. Figure 6 shows that by increasing the value of α to 2.0, the congestion widow for Paced-Westwood increases at a much faster rate than with α set to 1.0. We compared the performance of α-paced Westwood with Paced-Newreno. Figure 7 shows this evaluation plot. We see that α-paced Westwood performs marginally better than Paced-Newreno.

C. Multiple Flows

In light of the investigation of [2] we must evaluate the performance of our enhancement to see if the synchronisation effect of pacing is exacerbated or diminished by the bandwidth estimation procedure of TCPW. In the experi-

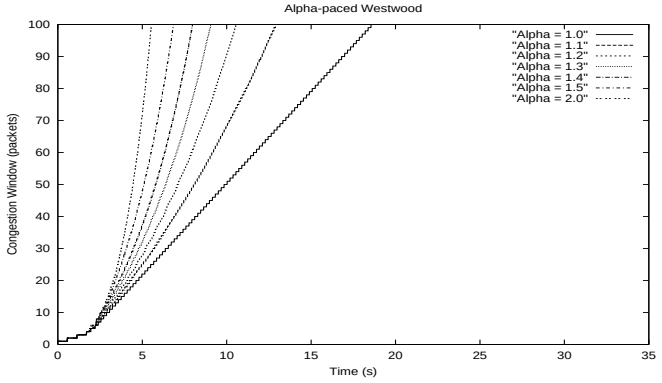

Fig. 6. *Performance of Alpha-paced Westwood*

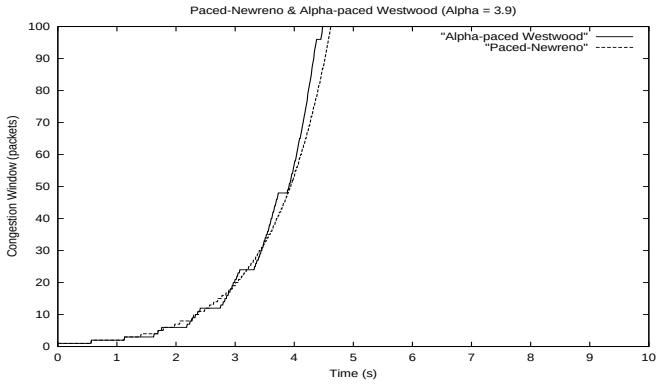

Fig. 7. *Alpha-paced Westwood vs Paced-Newreno*

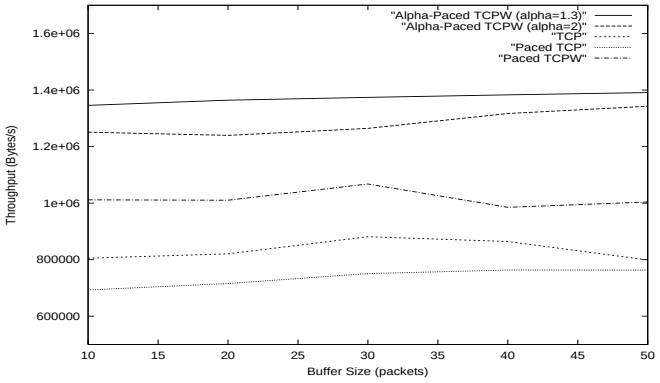

Fig. 8. *Single Flows*

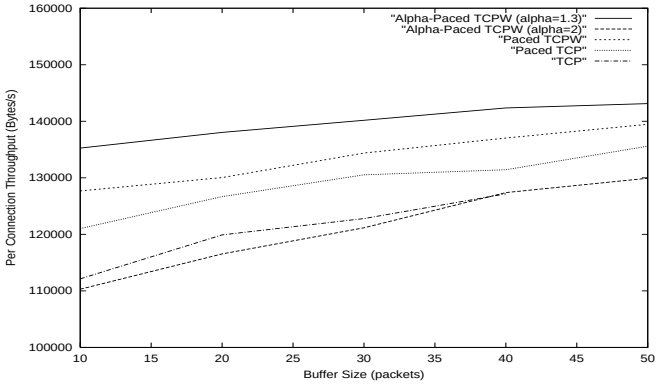

Fig. 9. *Multiple Flows: 10 connections, bandwidth*delay=200 packets*

ment we observe that for buffer sizes which are of the order of one fourth of the bandwidth * Delay product Figure 9, α-paced Westwood outperforms the other variants when there are multiple flows as well.

VI. Conclusions and Future Work

We analyzed the effect of small buffers on TCP throughput. We recognized TCP pacing as a potential solution. We also noted that the calculation of packet inter-departure time in the current attempts, blindly use the instantaneous TCP sending rate. Not using TCP's sending rate as the dictating metric, we instead make use of the bandwidth-share estimate (BSE), maintained by TCP Westwood to set the pacing interval. We call this newly proposed protocol *Paced-Westwood*. We also realize that we need to scale BSE further to mitigate the effects of positive feedback in BSE. For this, we propose another protocol that we call α-paced Westwood. This uses a scaling parameter α to enforce convergence of BSE and the pacing interval.

Our work evaluated the performance of TCP variants in a single flow case, as well as the multiple flows case. Our α-paced Westwood outperforms other variants in the small-buffer scenarios.

References

[1] ACK spacing for high bandwidth-delay paths with insufficient buffering. *Internet Draft draft-rfced-info-partridge-01.txt*, September 1998.

[2] Amit Aggarwal, Stefan Savage, and Thomas Anderson. Understanding the performance of TCP pacing. In *INFOCOM (3)*, pages 1157–1165, 2000.

[3] Chadi Barakat and Eitan Altman. Analysis of the phenomenon of several slow start phases in TCP. *ACM Sigmetrics*, 2000.

[4] Chadi Barakat, Nesrine Chaher, Walid Dabbous, and Eitan Altman. Improving TCP/IP over geostationary satellite links. *IEEE Globcom*, 1999.

[5] R. Lo Cigno, G. Procissi, and M. Gerla. Sender-side TCP modifications:an analytic study. *Networks*, 2002.

[6] M. Gerla, M. Y. Sanadidi, R. Wang, A. Zanella, C. Casetti, and S. Mascolo. TCP Westwood: Congestion window control using bandwidth estimation. *IEEE GlobeComm*, 2001.

[7] John Heidemann and Visweswaraiah Vikarm. Performance interactions between P-HTTP and TCP implementations. *ACM Computer Communication Review*, 27, 1997.

[8] Van Jacobson. Congestion avoidance and control. In *ACM SIGCOMM '88*, pages 314–329, Stanford, CA, August 1988.

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

[10] Steve McCanne and Sally Floyd. Ns-simulator. 1995.

[11] Craig Partridge, Joanna Kulik, Robert Coulter, and Denis Rockwell. Paced TCP for high bandwidth-delay networks. *WOSBIS*, 1999.

[12] Gupta Rajshri, Mike Chen, Steven McCanne, and Jean Walrand. Webtp:a receiver-driven web transport protocol. *INFOCOM*, 1999.

[13] A. Zanella, G. Procissi, M. Gerla, and M. Y. Sanadidi. TCP Westwood: Analytic model and performance evaluation. *IEEE GlobeComm*, 2001.