Burst TCP: an approach for benefiting mice flows

Glauco E. Gonçalves[1], Stenio Fernandes[2,1], Denio Mariz[3,1], Djamel Sadok[1]

[1]Universidade Federal de Pernambuco – UFPE
Caixa Postal 7851 – 50732-970 – Recife – PE

[2]Centro Federal de Educação Tecnológica de Alagoas – CEFET-AL
Rua Barão de Atalaia, s/n – 57020-510 – Maceió, AL

[3]Centro Federal de Educação Tecnológica da Paraíba – CEFET-PB
Rua 1º de Maio, 720 – 58015-180 – João Pessoa, PB

{glauco, stenio, denio, jamel}@gprt.ufpe.br

Abstract. *Standard TCP congestion control mechanisms degrade performance of small flows, especially during the Slow Start phase, which often causes multiple packet losses. We propose a modified TCP startup mechanism, called Burst TCP (B-TCP), which employs a responsive growth scheme based on current window size, to improve performance for small flows. Our simulation experiments, considering heavy-tailed traffic, show that B-TCP can significantly reduce both transfer times and packet losses for small flows without causing damage to large flows. Additionally, B-TCP is easy to implement and requires TCP adjustment at the sender side only.*

1. Introduction

The Transmission Control Protocol (TCP) is responsible for supplying reliable data transport service on the TCP/IP stack and for carrying most than 90% of all Internet traffic [Fraleigh et al. 2003]. In addition, the stability and efficiency of the actual TCP congestion control mechanisms have been extensively studied and are indeed well known by the networking community [Chiu and Jain 1989]. However, new Internet applications and functionalities continuously modify its traffic characteristics, demanding new research in order to adapt TCP to the new reality of the Internet. In particular, a traffic phenomenon known as "mice and elephants" has been motivating important researches around the TCP.

The mice and elephants metaphor alludes to a characteristic observed in traffic of heavy-utilized Internet links, where many flows (mice flows) carry few packets (or bytes), and few flows (elephants flows) carry many packets (or bytes). In some cases, for example, as little as 0.02% of all flows contribute with more than 59.3% of the total traffic volume [Mori et al. 2004].

It can be said that elephant flows are those that do not only contribute significantly to the total load, but also show long-lasting duration. However, there is no consensus on an exact quantitative definition of what elephant or mouse flows are. Consequently, many network operators use their own criteria for establishing a definition. For example, a possible quantitative definition of an elephant flow is those that contribute with more of 0.1% of all sampled packets [Mori et al. 2004].

In this scenario, the main point is that the TCP was designed for elephants. This causes several problems for mice flows forcing it to achieve poor performance. One well-known problem is that the initial and final TCP phases represent a significant amount of the life time for small flows, and cause minor overhead for large flows [Padmanabhan and Mogul 1994]. Moreover each TCP flow examines the available bandwidth in an independent way, even if other concurrent flows exist in the path. Therefore each new flow is forced to run the Slow Start.

The Slow Start algorithm initiate with a small congestion window (*cwnd*) of typically one segment, and each new acknowledgement (ACK) is an explicit permission for sending two new packets [Chiu and Jain 1989], which causes the known exponential behavior. However, at Slow Start phase, TCP uses little network bandwidth and small flows do not achieve optimal performance. Moreover, this slowness is one of the causes of insufficient data for activate Fast Retransmit/Fast Recovery, and thus, Retransmission Time Out (RTO) is the only mechanism for loss detection, increasing small flows latency because it higher initial estimative.

While the first problem is based in the slow increase of the exponential algorithm, the second problem is consequence of its increasing rule, which states that Slow Start must duplicate its *cwnd* at each round trip time (RTT). This rule associated with the use of default parameters at the beginning of transmission, often cause a severe buffer overflow at the bottleneck link. This buffer overflow results in multiple packet losses, which lowers the aggregate throughput and forces TCP performance to degrade substantially.

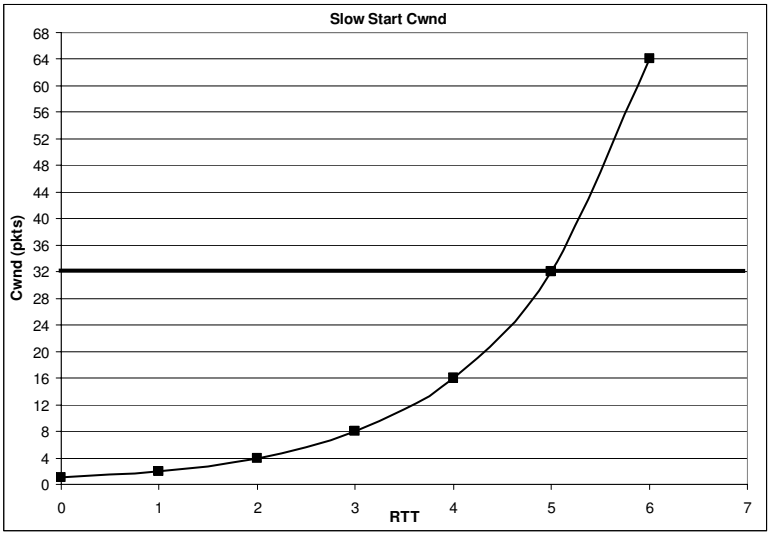

Figure 1. Example of bad bandwidth estimative of Slow Start

Figure 1 shows this problem. Please assume that the maximum number of packets accepted on the network is 32, indicated by the bolded line, and the Slow Start threshold (*ssthresh*) variable is set to 64 packets or more. While *cwnd* is below of 32 no problem happens, but when *cwnd* reaches 32 (in the fifth RTT) and new packets start to be acknowledged problems came arise. When all packets are acknowledged the cwnd reaches 64 packets. Thus, the number of sent packets in next RTT will be 64 packets, but as the available bandwidth is 32 packets, about 50% of all packets will be lost.

This undesirable effect happens because Slow Start is "blinded" for its own congestion contribution, i.e., Slow Start estimate that if a new ACK returns then more packets can be sent. But the truth is that increasing it window in this greedy way contributes to increase the number of packets in the network, increase the drop probability in bottleneck routers [Wang et al. 2004] and reducing the overall network throughput. As an aggravation, when this algorithm is applied to mice flows the problem is still bigger, because small flows depends of timeouts and will experience large delays to recover of these multiple losses [Wang et al. 2000].

The war between mice and elephants allied to TCP traffic prevalence on the Internet show the constant necessity to reexamine the TCP congestion control mechanisms, in order to adapt it to the new traffic characteristics emerging in the Internet. Thus, the objective of this work is present *Burst* TCP, B-TCP for short, an intuitive TCP modification, which changes the initial TCP growth scheme to improve small flows performance without causing damage to large flows.

The remaining of this paper is organized as follows. Section 2 discusses similar researches on adjusting TCP congestion control behavior, with emphasis on those that proposed modifications in the Slow Start phase. Section 3 presents the B-TCP, a new approach for the window growth scheme aiming to favor mice flows with minor impact on other flows. A simulative analysis of B-TCP is presented in Section 4 and results are discussed for various metrics. The paper is concluded in Section 5.

2. Related Work

TCP was first specified in [Postel 1981], despite of many later modifications [Braden 1989], four main algorithms used for congestion control are responsible for its general behavior: Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery.

The Slow Start algorithm aims at finding out the available bandwidth, assuming that there is no existing information regarding to the network conditions. Hence, it increases exponentially the transmission rate until a certain threshold is reached. Then, if no segment loss is verified, the Congestion Avoidance algorithm is activated, which increases linearly the transmission rate preventing packet drops. In both cases, any packet drop forces TCP to call the beginning of the Slow Start and to reduce the threshold.

The packet loss detection is done using a timer, named RTO: if no ACK is received after the time indicated by the RTO value, the lost packet is retransmitted. If the sender TCP receives three or more ACK for a given segment, it is assumed that a loss occurred for that segment, since those ACKs were triggered by subsequent segments arriving at the receiver. In this case, the Fast Retransmit algorithm is activated to retransmit the lost packet even before the RTO being reached. However, after the Fast Retransmit, the congestion control uses the Fast Recovery that, differently of the timer scheme, calls the Congestion Avoidance phase and not the Slow Start phase.

There is a considerable amount of research intending to solve this problem with the main goal to enhance the performance for small flows. Earlier research works proposed to maintain history information from previous flows to infer the network state. The Control Block Interdependence, described in [Touch 1997] and implemented in

[Balakrishnan et al. 1998], considers sharing state information contained in an already existent data structure: the TCP Control Block, which maintains RTT estimates and congestion control variables. Thus, each new TCP connection can receive a portion of the network resources based in the Control Block estimate. Balakrishnan et al. (1999) present the Congestion Manager (CM), whose idea is to maintain network statistics for all host flows, independently of transport protocol used.

Other works proposed modifying the standard Slow Start mechanism in order to achieve higher utilization during this initial phase. Allman et al. (2002) surveys works that enhance TCP performance increasing the initial window from one to four segments. These evaluations show that larger initial window scheme improves the throughput and transfer time of short-lived TCP in a variety of scenarios, such as satellite links and dialup modems links. However, in heavily congested scenarios where all flows use an initial window of four segments, the drop rate increases by 1% to 2%.

Smooth Start [Wang et al. 2000] address the aggressive behavior of the Slow Start growth, and proposes to alleviate the transition between the Slow Start phase and the Congestion Avoidance phase, creating a transitional phase called Smooth Start phase, where the *cwnd* grows with a lesser rate. Through simulations authors shown that it simple modification can significantly reduce packet losses and produce less bursty traffic than slow start.

In this same line, Wang et al. (2004) propose a modification in the Slow Start mechanism, called Adaptive Start (Astart), which uses the Eligible Rate Estimate (ERE) mechanism employed in TCP Westwood [Mascolo et al. 2000] for setting the value of the slow start threshold (*ssthresh*) variable. Thus, when *ssthresh* is reset to a value higher than the current one, the *cwnd* grows exponentially to the new value. This mechanism avoids overestimates as well as underestimates of available bandwidth.

Gallop-Vegas mechanism [Ho et al. 2005] aims at solving a problem of TCP Vegas [Brakmo and Peterson 1995], which changes too early from Slow Start to Congestion Avoidance phases. Gallop-Vegas mechanism tries to reduce the burstiness, to raise the rate to the available bandwidth in shorter time, and to improve the start-up performance. Such an approach puts the *cwnd* increasing phase between the exponential growth and the linear growth.

The TCP Smart Framing (TCP-SF) [Mellia et al. 2005] enhances TCP performance in the operating region where the *cwnd* is small. A sender host using this algorithm is allowed to send four small segments, whose aggregate payload is equal to the initial *cwnd*. Thus, the sender will have a better chance to infer loss with Fast Retransmit, and without increasing heavily the network load. Despite the better small flows performance achieved, Iyengar et al. (2003) argue, based on the heavy-tailed behavior of Internet traffic, that TCP-SF causes an increased amount of ACKs in the reverse path.

The technical report of Iyengar et al. (2003) discusses some of these proposals, and suggests a classification scheme with three categories: proposals that reduce the overhead of initialization or finalization connection phases; proposals that employ mechanisms to share network state; and proposals that improve Slow Start performance. Moreover, this work proposes some guidelines to compare and evaluate these proposals.

Thus, a new TCP start scheme is acceptable due to the fact that the cited proposals have some limitations. Smooth Start solves the Slow Start problem of multiple packet losses, but it does not solve the initial slow increase problem, that is, it does not focus on the problem of mice flows. The TCP-SF activates the Fast Retransmit in cases of loss, but maintains both problems of Slow Start (few initial packets and multiple losses). The Astart and the Gallop-Vegas solve these two problems, but both proposals use bandwidth estimation schemes that can be inaccurate due to route changes, for example.

3. The Burst TCP

In this section we present B-TCP, a new window growth approach during the TCP start phase, which aims to favor mice flows with minor impact on elephants. As explained in Section 1, Slow Start presents some problems associated with it exponential behavior. One of them is that it initiate with a very low rate, even increasing it very faster. Hence, firstly, we developed a simple Slow Start generalization, which maintains the exponential growth of Slow Start but modifies it exponential base.

Jin et al. (2004) explained that current TCP congestion control research follow an important guideline: the congestion control behavior is designed at a flow level and, after this, the packet level implementation is considered. The first approach models the macroscopic constraints of the flow, such as achieved throughput and fairness. On other hand, the packet level specification is determined by the constraints and objectives described at flow level.

Analyzing the congestion window dynamics of Slow Start in the packet level, we understand that *cwnd* increases by one packet for each ACK arrival. Thus, disregarding the effect of some issues such as delayed ACKs receivers and the topological diversity of the Internet, we have that for each ACK arrival

$$cwnd \leftarrow cwnd + 1$$

but in the flow level, *cwnd* grows exponentially in the base 2

$$cwnd \leftarrow 2^{RTT}.$$

From the flow level model we generalize the Slow Start algorithm as below,

$$cwnd \leftarrow b^{RTT}$$

where *b* is the base. The packet-level scheme becomes

$$cwnd \leftarrow cwnd + (b-1).$$

Figure 2 compares the congestion window dynamics of Slow Start and it generalization using a base equal to 3. Please note that the generalization increases the window growth and consequently it improves the individual transfer time for small flows when there is available bandwidth.

However the generalized Slow Start shows the fail of the exponential growth behavior. For instance, from Figure 1, one can note that Slow Start achieves the *cwnd* of 64 packets in six RTTs. But, using the base equal to 4, it reaches the same value in just three RTTs. This means that generalized form of Slow Start anticipates network congestion, since while Slow Start sends 95 packets before occur losses it generalized

form sends only 53 packets, thus, considering a file size of 64 packets, Slow Start could terminate the transfer without retransmissions.

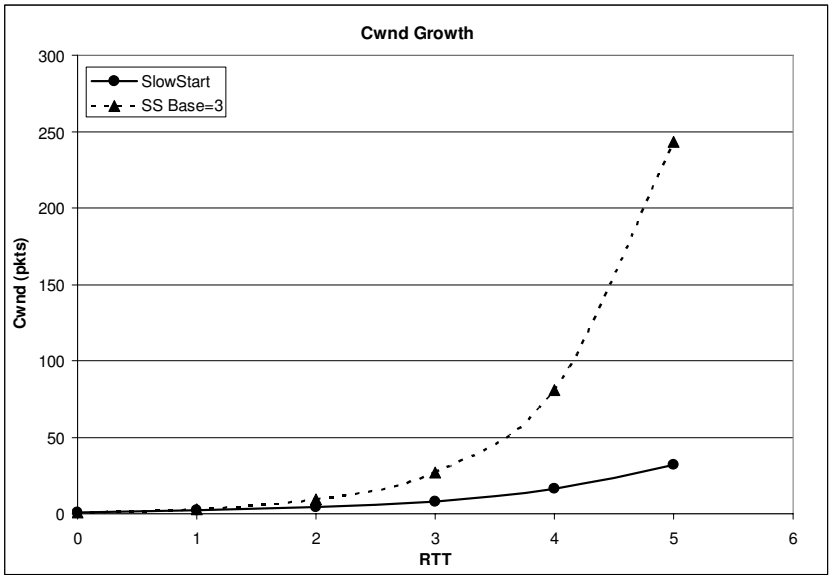

Figure 2. Slow Start and generalized Slow Start congestion window dynamics

This problem in the exponential design motivated us to propose a second algorithm for reduce the two Slow Start problems, called Burst TCP.

Please note that the function of Slow Start is increasing the TCP rate (or congestion window, in practical terms) to reaches the available link bandwidth. So we can metaphorically say that this process looks like the task of fill a pipe or a bottle with water. However, Slow Start approach is counter-intuitive to the standard manner to do this task [Wang et al. 2000]. Indeed, to do this we must initiate pouring out water in the bottle quickly and reducing it progressively to avoid spill out water. B-TCP follows this concept, that is, at initial RTTs, the algorithm increase the congestion window through large steps and reduces these steps gradually.

In this way, the B-TCP proposal is to create a window growth scheme inversely proportional to the window size. The intuition behind this scheme is that if a flow increases it window size then it has more data to send, and if this flow grows then it leaves the mice class and turns an elephant. Thus, this proposal employs the mice threshold (*mice_t*) for indicate flow's class change.

To obtain the desired dynamics, the flow level behavior was first defined as:

$$cwnd \leftarrow cwnd + floor\left(\frac{mice\_t - cwnd}{f}\right)$$

where *f* is the smooth-out factor responsible to control the aggressiveness of B-TCP(*f*), and the *floor* function returns the largest integer less than or equal to *mice_t – cwnd*. The threshold, called mice threshold (*mice_t*), indicates whether a flow has been changed from a mouse to a small elephant, that is, when the window reaches this value the flow can not be considered a mouse because it already have data to send. Thus, the packet level implementation was designed as:

$$cwnd \leftarrow cwnd + \frac{floor\left(\frac{mice\_t - cwnd}{f}\right)}{cwnd}.$$

The idea of this implementation is that each increment in *cwnd* will be a fraction of *cwnd*, guaranteeing the dynamic designed in the flow level specification. Figure 3 compares the congestion window dynamics at flow level of Slow Start, B-TCP(4) and B-TCP(8) with *mice_t* equal to 32 packets. Please note as behave the packet-level implementation.

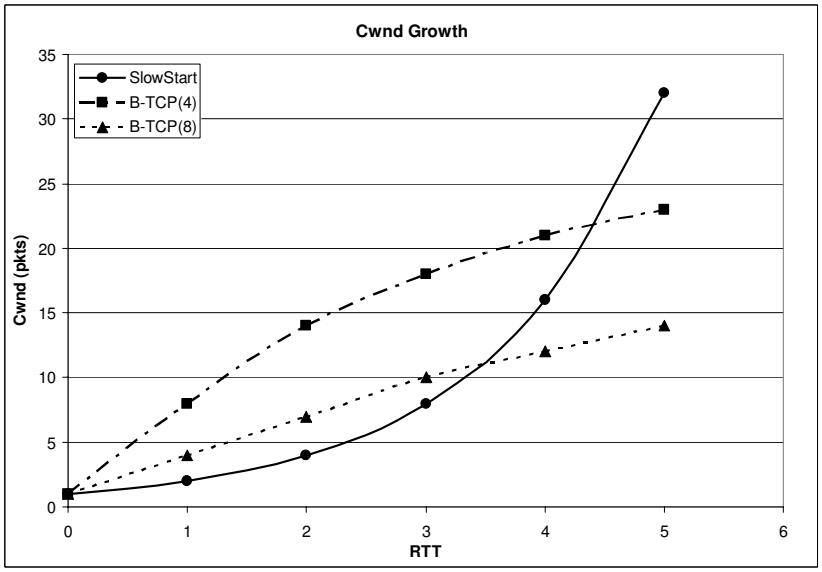

Figure 3. Slow Start, B-TCP(4) and B-TCP(8) congestion window dynamics

B-TCP has other parameter to be controlled beyond the mice threshold. The smooth-out factor *f* governs the increase window value, and a priori it value is constant and set arbitrarily. Moreover, other B-TCP modification is that a flow using it, must maintain it dynamics up to *cwnd* reaches a certain threshold or until the connection detect a loss, behaving as the standard Reno TCP, from this point and so on. The loss detection, on other hand, show that network is congested and therefore B-TCP must turn off it aggressive behavior.

Figure 4 shows the pseudo-code of B-TCP and illustrates it behavior in relation to the main TCP events: ACK arrivals, three duplicated ACKs detection, and timeout expiration. Others TCP pieces of code had not been shown directly, but are represented by the `Standard TCP` indication. The algorithm considers *cwnd* in packets, following the scheme used by the NS-2 simulator [NS-2 2007].

The `btcp_enabled` boolean variable implements the B-TCP turn off, which is set to FALSE when a loss is detected and remains with this value forever, i.e., the TCP protocol does not perform B-TCP algorithm ever again during the flow's lifetime. Using this strategy B-TCP can avoid other losses caused by its aggressive growth. On the other hand, this strategy can be uninteresting for wireless environments where packet losses may be caused by transmission errors. However, the scenarios studied in this paper are limited to wired networks.

Another observation in this pseudo-code is that, in some cases, the floor function can set the increment variable to zero, therefore this pseudo-code employs a condition to avoid that the congestion window obtains no increment. Moreover, such modification, jointly with the inversely proportional window growth of B-TCP, allows a smooth transition between the initial phase and the congestion avoidance phase.

```
When non-duplicated ACK arrival:
If (cwnd < ssthresh) {
     If (btcp_enabled==TRUE) {
      increment = floor((mice_t - cwnd)/f)
      If (increment==0) increment = 1
      cwnd += increment/cwnd
    } else { cwnd += 1  }
}else { //go to Standard TCP }
When 3 Duplicated ACK or when Timeout expires
btcp_enabled=FALSE
//go to Standard TCP
```

Figure 4. Pseudo-code of B-TCP(*f*) with mice threshold *mice_t*

4. Simulation Results and Discussion

We carried out simulations using NS-2 to study the effect of B-TCP for mice traffic competing with elephant traffic.

4.1. Simulation Scenario

The simulation uses a dumbbell topology, as depicted in Figure 5, where each side has five nodes. The flow direction is left to right with ACKs in the opposite way. Each node has a variable number of TCP flows (50 to 250 flows per node).

The peripheral links bandwidth is 10 Mbps and its one-way delay is 5 ms, for the bottleneck these values are 1.5 Mbps and 35 ms, respectively. The queues sizes on each router were set equal to the bandwidth-delay product (BDP) of an individual link. Thus, with packets of 1024B, the queue size value was set to 7 packets.

Each flow is a FTP transfer whose file size (in packets) follows a Pareto distribution with the scale parameter is 4 and the shape parameter is 1.2. The chosen scale value limits the lower bound of the Pareto distribution to 4 packets (4 KB), which is interesting because, for any TCP flavor (Newreno or B-TCP), size files below 4 packets, in loss cases, do not activate the Fast Retransmit. The shape value (1.2) was chosen based on literature information [Carofiglio et al. 2007].

In order to avoid synchronization among flows, the initial time of each flow is chosen from a uniform distribution between 0 and 900 seconds. The total simulation time is set to 1000 seconds. Moreover, each flow is grouped in the mice or elephant classes according to a classification threshold. In this experiment, two different thresholds have been used. First, the classification threshold was set equal to 32 KB because this value is the mice threshold used on B-TCP. The other classification

threshold was chosen using the AEST method, which is a non-parametric statistical method created by Crovella and Taqqu (1999) for estimate the tail index of a heavy-tail distribution from empirical data. This method can be used to calculate the first point in the tail of the distribution, that is, the threshold between mice and elephants. Thus the calculated threshold was 82 KB.

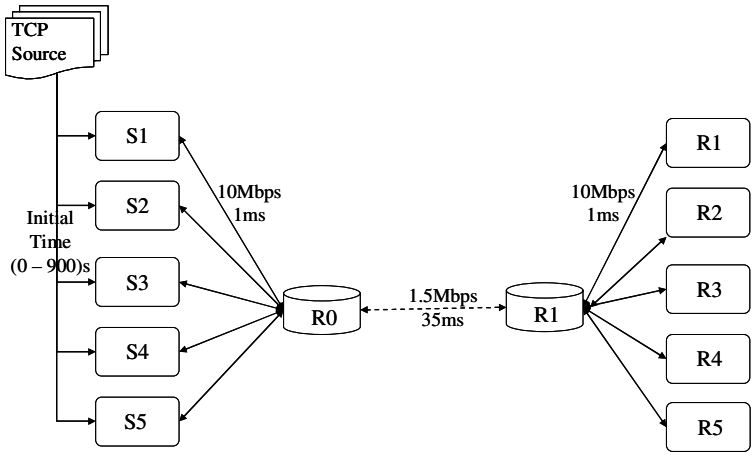

Figure 5. Network topology for simulation

In each simulation, the number of TCP sources varies. Each case is repeated 50 times to guarantee results that are statistically acceptable. And all results use a 95% confidence level.

4.2 Simulation Results

The first issue to be answered is how to determine the number of flows that causes congestion. Preliminary study demonstrates that the typical bottleneck link utilization is obtained for 250 flows and 1250 flows with Newreno TCP Flavor. For few flows (250 flows) the mean link utilization is about 2.8%; however there are some peaks with 1 to 3 seconds duration where the utilization reaches between 90% and 100%. In the other case (1250 flows) the mean link utilization obtained was around 19.6% with link utilization peaks of up to 85% lasting as long as one minute.

The values of the transfer time were computed for each class. Figure 6 shows the mean transfer time (in seconds) in a line graph for the mice class using different values as mice threshold. Considering Figure 6(a), a first observation is that the small confidence interval guarantee reliability, in this case the accuracy is about 5%. Moreover, B-TCP(4) and B-TCP(8) obtain smaller mean values than Newreno, the difference is about 38% in the light congested case and 19% in the heavy congested case. This occurs because B-TCP considers its own contribution for congestion, and reduces its growing rate as the congestion window increases.

Please note that the choice of the smooth-out factor affects the transfer time. At best B-TCP(4) improves the transfer time by approximately 18%. Despite this with 1000 and 1250 flows nothing can be said because of confidence intervals overlap. A second observation is that the B-TCP(4) line slope is greater than that of B-TCP(8) and Newreno, hence indicating that the difference between them can diminish with the increase of the number of flows.

Therefore, we run this case for 1500, 1750, 2000, 2250, and 2500 flows, and our previous suspicion was confirmed: B-TCP(4) exceeds B-TCP(8) and reaches Newreno, but B-TCP(8) continues improving Newreno transfer times despite congestion increase, for instance, for 2250 flows the improvement is approximately 10%. For 2500 flows nothing can be said because confidence intervals overlap.

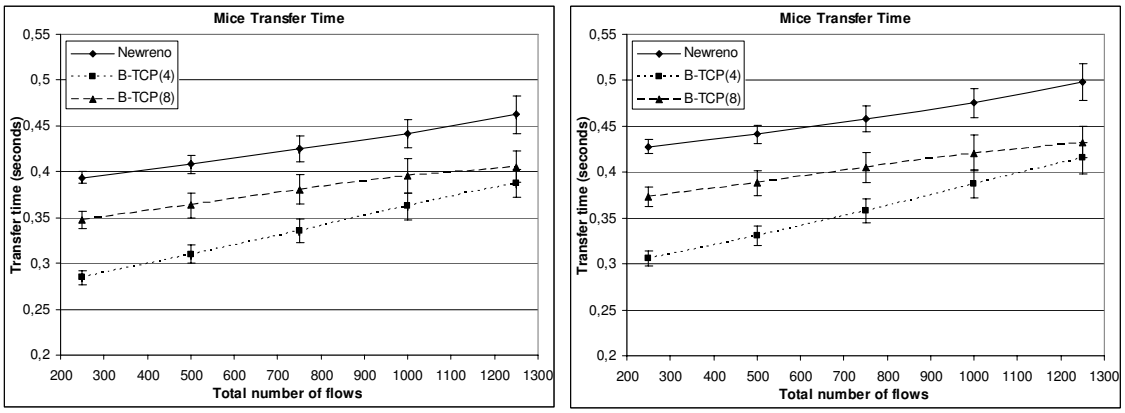

Figure 6. Transfer time graph for mice class using (a) 32Kb as threshold and using (b) AEST threshold

Figure 6(b) shows transfer time results using AEST method in calculus of the threshold between mice and elephants. Please note that the linear behavior for each TCP flavor is the same that the one in Figure 6(a), despite its plotted values being different. Such result shows that our rule of thumb choice of 32KB is a good value for representing the threshold.

A second metric evaluated in this experiment was the percentage of lost packets for each class. Figure 7 shows packet loss observed, which allow some comparisons. In these figures, the numbers 1, 2, 3, 4, and 5, are related to 250, 500, 750, 1000, and 1250 flows, respectively.

The first observation on this metric is that the percentage of lost packets in the mice class for B-TCP(4) is many times bigger than B-TCP(8) or Newreno. On the other hand, B-TCP(8) achieves lower values than Newreno. This occurs because B-TCP initiates its connection with a bigger packet burst which can increase the number of lost packets. For instance, consider the bottleneck BDP and buffer of this experiment, whose values are 7 packets, and only two B-TCP(4) flows. After the first packet arrival of each flow, B-TCP(4) increases its congestion window to 8 packets, therefore the total number of packets in network will be greater than the BDP and the bottleneck buffer, causing lost of two packets. Thus, even with few concurrent flows, B-TCP(4) is more likely to lose it packets by routers drop.

The second interesting observation regarding the packet loss metric is relative to elephant graph. This graph shows that B-TCP algorithms reduce the mean number of lost packets for elephants. Please note that, for B-TCP(8) results, absolute values for elephants are, surprisingly, smaller than absolute mice values. In this line, the last observation is that the overall number of lost packets for B-TCP(8) is less than that of other cases, since it finishes its transmission before the sender rate reaches the available bandwidth therefore reducing the loss probability.

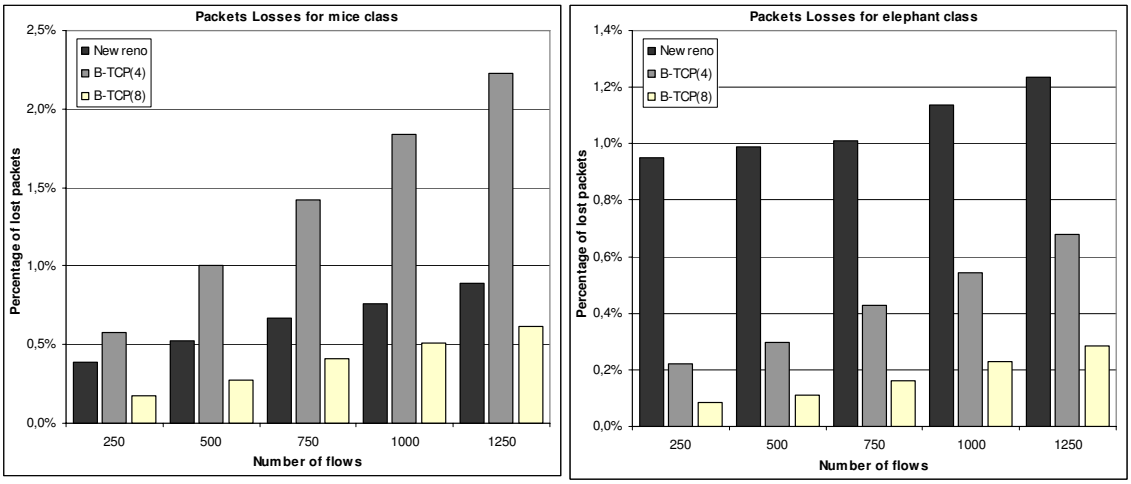

Figure 7. Packet losses graphs

Looking for transfer time metrics only, B-TCP(4) obtained better performance than B-TCP(8). But, this last metric has shown that B-TCP(4) has a greedy behavior which can achieve poor results in networks with high congestion level. Therefore we run the same case using a 1.0 Mbps bottleneck link. The idea here is also to increase the congestion and, at the same time, modify router buffers, which was set to only 4 packets.

Figure 8 presents the mice transfer time for Newreno, B-TCP(4) and B-TCP(8) TCP flavors, and shows clearly that B-TCP(4) achieves poor performance in relation to Newreno and B-TCP(8). The latter, on the other hand, continues achieving good results.

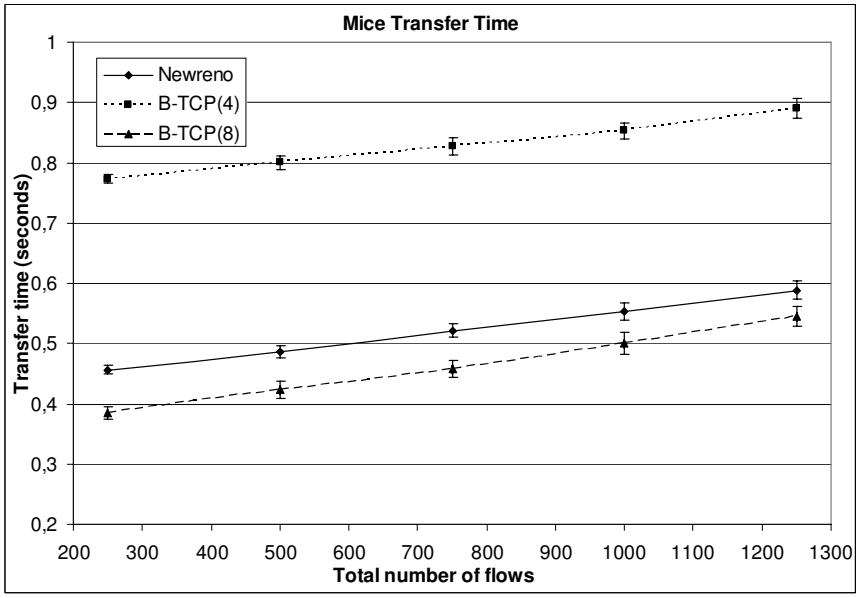

Figure 8. Transfer time graph for mice class and 1Mpbs link rate

We still evaluated the effect of the buffer size. First we evaluated the effect of use RTT to calculate the buffer size. Thus, buffer size was set to 14 packets. Figure 9 shows result for the transfer time of the mice class using this rule. As expected, B-TCP still achieves good performance, and the use of a large buffer improves it transfer time

reaching about 30% in the better case. Moreover, the difference between Newreno and B-TCP(4) increased to approximately 43%.

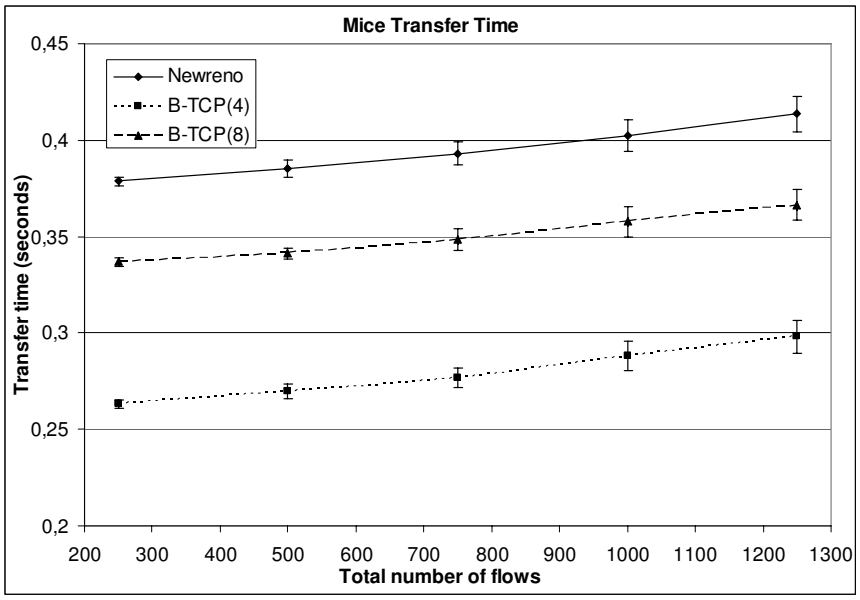

Figure 9. Transfer Time graph for mice class using augmented buffer

We also evaluated B-TCP using a small buffer size. Based on [Appenzeller et al. 2004], we calculated the buffer size using $(RTTxC)/\sqrt{n}$, where n is the number of elephant flows in the set, and C is the link capacity. The number of elephant flows is about 10% of all flows therefore the buffer was set to 3 packets. Figure 10 shows result for this evaluation. In this case, B-TCP achieves poor results than Newreno due the bursty behavior of B-TCP. However, this bad result can be explained by the simplicity of our topology. Since it is known that the rule must be employed when there are a high number of flows through the link.

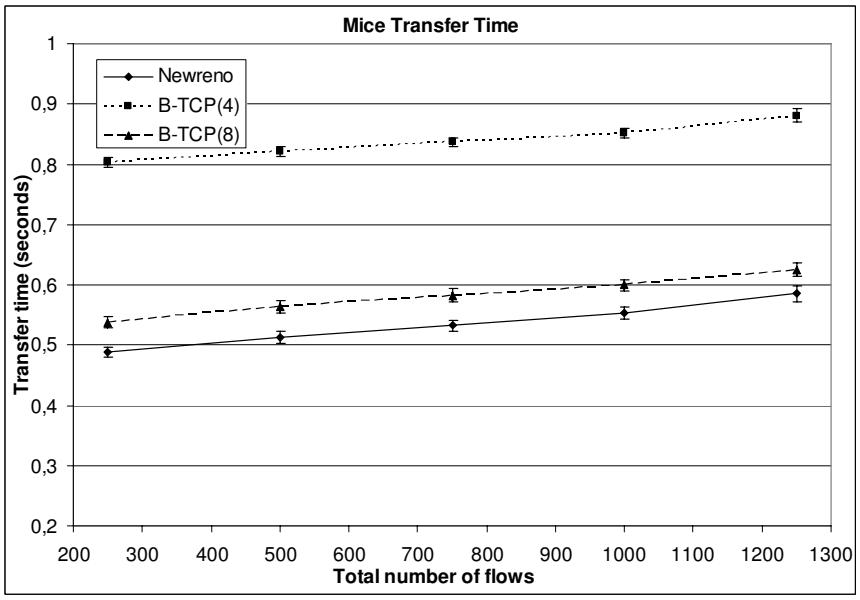

Figure 10. Transfer Time graph for mice class using reduced buffer

Other scenarios and simulation parameters were also used but the results were quite similar to the results showed above. Thus, due to space restriction, we have chosen some representative scenarios to make the B-TCP explanation easier and to verify the B-TCP efficiency.

5. Conclusions

This work proposed a new algorithm to start TCP connections, called B-TCP, which is an intuitive modification, easy to implement and that requires TCP adjustment at the sender side only. We evaluated performance of B-TCP in a scenario with elephant and mice flows competing on a congested network. The heavy-tailed traffic was generated following a Pareto distribution and its parameters were obtained from the literature.

B-TCP, in a general way, improves the transfer time for mice flows and, in some cases, for elephant flows also. Moreover, results showed that under high congestion, B-TCP performance for mice flows is quite similar to that of Newreno. However, in our experiments it was not possible to determine, with confidence, how much the use of B-TCP harms elephant flows in this metric.

Two versions of B-TCP were evaluated, using different smooth-out factor: 4 and 8. The former proved to achieve the better transfer time values, but it increases packet loss for mice flows, affecting it performance in heavy congested links. The effect of buffer size also was evaluated showing that B-TCP performance is sensible to this parameter.

Future work of this proposal includes considering other topologies and link types. Moreover, the effect of the background traffic and the reverse traffic will be evaluated.

References

Allman, M., Floyd, S. and Partridge, C. (2002) “Increasing TCP’s Initial Window”. RFC 3390.

Appenzeller, G., Keslassy, I., and McKeown, N. (2004) “Sizing Router Buffers”. ACM SIGCOMM 2004, Portland, USA.

Balakrishnan, H., Padmanabhan, V., Seshan, S., Stemm, M. and Katz, R. (1998) “TCP Behavior of a Busy Internet Server: Analysis and Improvements”. Proceedings of INFOCOM.

Balakrishnan, H., Rahul, H. and Seshan, S. (1999) “An Integrated Congestion Management Architecture for Internet Hosts”. Proceedings of SIGCOMM.

Braden, R. (1989) “Requirements for Internet Hosts -- Communication Layers”. RFC 1122.

Brakmo, L. and Peterson, L. (1995) “TCP Vegas: End-to-end Congestion Avoidance on a Global Internet”, IEEE J. Sel. Areas Commun., VOL. 13, NO. 8, pp. 1465-1480.

Carofiglio, G., Garetto, M., Leonardi, E., Tarello, A. and Marsan, M. (2007) “Beyond fluid models: Modelling TCP mice in IP networks under non-stationary random traffic”. To appear in: Computer Networks, v. 51, i. 1, pp. 114-133, January 2007.

Chiu, D. and Jain, R. (1989) "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks". Computer Networks and ISDN Systems, n. 17, pp. 1-14.

Crovella, M., and Taqqu, M. (1999) "Estimating the Heavy Tail Index from Scaling Properties". Methodology and Computing in Applied Probability.

Fraleigh, C., Moon, S., Lyles, B., Cotton, C., Khan, M., Moll, D., Rockell, R., Seely, T. and Diot, C. (2003) Packet-level traffic measurement from the Sprint IP backbone. IEEE Network Magazine, v. 17, n. 6, pp. 6-16.

Ho, C.-Y., Chan, Y.-C., and Chen, Y.-C. (2005) "An enhanced slow-start mechanism for TCP Vegas". 11th International Conference on Parallel and Distributed Systems, v. 1, pp. 405-411.

Iyengar, J., Caro, A., and Amer, P. (2003) "Dealing with Short TCP Flows: A Survey of Mice in Elephant Shoes". Technical Report, University of Delaware.

Jin, C., Wei, D. and Steven, L. (2004) "FAST TCP: motivation, architecture, algorithms, performance". INFOCOM 2004, Hong Kong, pp. 2490-2501.

Mascolo, S., Casetti, C., Gerla, M., Lee, S. and Sanadini, M. (2000) "TCP Westwood: congestion control with faster recovery". UCLA, Technical Report #200017.

Mellia, M., Meo, M. and Casetti, C. (2005) TCP smart framing: a segmentation algorithm to reduce TCP latency. IEEE/ACM Transactions on Networking, v. 13, pp. 316-329.

Mori, T., Uchida, M., Kawahara, R., Pan II, J. and Goto, S. (2004) "Identifying elephant flows through periodically sampled packets". Proceedings of Internet Measurement Conference, pp. 115-120.

NS-2 Network Simulator (2007) LBL, URL: http://www.isi.edu/nsnam/ns/. Visitado em: abril de 2007.

Padmanabhan, V. and Mogul, J. (1994) "Improving HTTP Latency". 2nd International World Wide Web Conference.

Postel, J. (1981) "Transmission Control Protocol". RFC 793, IETF.

Touch, J. (1997) "TCP Control Block Interdependence". RFC 2140, IETF.

Wang, H., Xin, H., Reeves, D., and Shin, K. (2000) "A simple refinement of slow-start of TCP congestion control". Fifth IEEE Symposium on Computers and Communications, pp. 98-105.

Wang, R., Pau, G., Yamada, K., Sanadidi, M., and Gerla, M. (2004) "TCP Startup Performance in Large Bandwidth Delay Networks". INFOCOM 2004, Hong Kong, pp.796-805.