The Effects of Fairness in Buffer Sizing

Mei Wang, Yashar Ganjali

Department of Electrical Engineering, Stanford University
{wmei, yganjali}@stanford.edu

Abstract. Buffer sizing in Internet routers is a fundamental problem that has major consequences in the design, implementation, and economy of the routers, as well as on the performance observed by the end users. Recently, there have been some seemingly contradictory results on buffer sizing. On the one hand, Appenzeller *et al.* show that as a direct consequence of desynchronization of flows in the core of the Internet, buffer sizes in core routers can be significantly reduced without any major degradation in network performance. On the other hand, Raina and Wischik show that such reduction in buffer sizing comes at the cost of synchronization and thus instability in the network.
This work unifies these results by studying the effects of fairness in buffer sizing. We show that the main difference arises from the implicit assumption of fairness in packet dropping in the latter result. We demonstrate that desynchronization among flows observed by Appenzeller *et al.* is caused by unfair packet dropping when a combination of TCP-Reno and the drop-tail queue management is used. We also show that bringing fairness in packet dropping will introduce synchronization among flows, and will make the system unstable as predicted by Raina and Wischik. Our analysis suggests that there is an intrinsic trade-off between fairness in packet drops and desynchronization among TCP-Reno flows when routers use the drop-tail queue management. Achieving fairness, desynchronization, small buffer size, and 100% link utilization at the same time is desirable and feasible yet challenging. The studies in this paper provide insights for further explorations in reaching this goal.

1 Motivation and Introduction

There have been increasing amount of interests on buffer sizing due to the important role that buffer plays in routers and the performance of the Internet. The goal of buffer sizing is to find out how small we can make Internet router buffers without any degradation in network performance. A plethora of recent work emerged to reduce buffer sizes [1–5] and to understand the relationships between buffer sizing and other parameters of the network [8–14], such as, throughput, delay, loss, stability [6, 7], and the impacts of various traffic conditions [8, 15].

Recently, there have been some seemingly contradictory results on buffer sizing in Internet core routers. Appenzeller *et al.* show that buffer sizes in core routers can be reduced significantly, without any major degradation in network performance [1], whereas, Raina and Wischik show that such reduction in buffer sizing can cause instabilities in the network [7]. Instability, here, is defined as the periodic variations in the aggregate congestion window size of the flows.

Which result is correct? To answer this question, we studied the dynamics of the system in terms of buffer sizing through mean-field theory [16, 22] analysis and *ns2* [17] simulations.

We demonstrated that there is an intrinsic trade-off between fairness among TCP-Reno flows and desynchronization among them using the drop-tail queue management scheme: fairness in packet drops can create synchronization among flows; conversely, unfair packet drops can lead to reduced synchronization.

Fairness has always been considered to be a desirable property in the network. The network is expected to treat individual flows in a fair manner when resources are limited. Synchronization among TCP flows, on the other hand, has always been considered to be an undesirable effect. Synchronized flows need much larger buffer sizes in core routers and cause local/global instabilities in the system, as well as degradation in the performance observed by individual flows.

To see the origins of the trade-off between fairness and desynchronization, let us consider a congested link in a network carrying a large number of flows. An Active Queue Management (AQM) scheme that fairly drops packets at times of congestion will impact a large percentage of flows since it will distribute packets dropped among the flows. On the other hand, an unfair AQM scheme can drop a lot of packets from a few flows, thus reducing the number of flows which see one or more packet drops. TCP-Reno flows react dramatically to packet drops by halving their congestion window sizes for each dropped packet [18]. Therefore, when a large number of flows see packet drops around the same time they will synchronously react by reducing their congestion window sizes. This may lead to a significant reduction in instantaneous throughput of the system. In an unfair AQM scheme, however, only a few flows will take the hit, and thus only a small fraction of flows will react to packet drops. Therefore, the aggregate congestion window will change less significantly.

Our results unify the seemingly contradictory conclusions mentioned above. Appenzeller *et al.* have shown that due to the desynchronization of flows in the core of the Internet, one can reduce buffer sizes in core routers by a factor of $\sqrt{N}$ from the original value of bandwidth-delay product [1]. Here N is the total number of flows going through the core router. Our analysis shows that this desynchronization is a direct result of the unfair nature of packet drops in TCP-Reno combined with the drop-tail queue management scheme (Section 2.1). Also, in Section 2.2 we show that introducing fairness in packet dropping using the drop-tail scheme creates global synchronization. This is consistent with Raina and Wischik's results. Our study shows that the results from both groups are correct, and the main difference can be well explained by understanding the effects of fairness in buffer sizing.

Fairness also explains the recent observation by Dhamdhere *et al.* that the majority of flows have dropped packets in a congestion cycle [8]. The same results are obtained in our work when there is fair packet dropping among flows. We demonstrate this through both the mean-field analysis[22] and *ns2* simulations (Section 2).

Achieving fairness, desynchronization, small buffer size, and 100% link utilization at the same time is desirable yet very challenging. The studies on the dynamics in this paper show insights and feasibility for further explorations in reaching this goal.

2 Fairness-Desynchronization Trade-off

In this section we study the relationship between synchronization and fairness in a network using TCP-Reno as the congestion control mechanism. We show that, when the congested routers use the drop-tail queue management scheme, there is an inherent trade-off between fairness in packet dropping and desynchronization amongst TCP-Reno flows. We investigate this trade-off through two different scenarios. First, we consider a scenario using the standard drop-tail queue management scheme in which TCP-Reno flows are desynchronized, and show that desynchronization comes at the cost of losing fairness in dropping packets. Second, we show that a slightly modified drop-tail queue management scheme in intermediate routers which is fair in dropping packets can lead to synchronization among TCP-Reno flows.

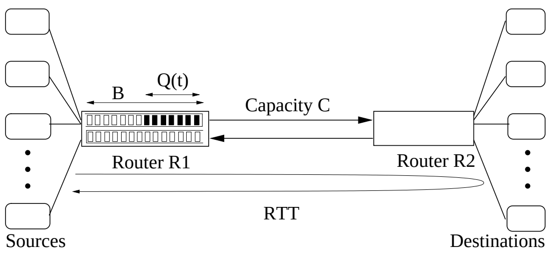

Fig. 1. Schematic of the model system considered. There are N long TCP flows connected to Router $R1$. The congested uplink from $R1$ to $R2$ is with capacity C. When the queue size $Q(t)$ becomes equal to buffer size B, newly arrived packets are being dropped based on the drop-tail algorithm.

2.1 Desynchronization is a result of unfairness

Recent results on buffer sizing in Internet core routers suggest that in a network carrying TCP-Reno flows and with the drop-tail queue management scheme in intermediate routers, buffer sizes can be reduced as the number of flows is increased [1]. More precisely, the amount of buffering needed in core buffer routers is inversely proportional to $\sqrt{N}$, where N is the number of flows going through the bottleneck link. The underlying assumption here is that as the number of flows N increases, flows become more desynchronized. Variations in individual congestion window sizes cancel each other out when the flows are desynchronized, and as a result, variations in the aggregate congestion window size are reduced, which means smaller buffers are able to accommodate the variations in aggregate congestion window size.

To further analyze these results, we use the same model as in Appenzeller's *ns2* simulations [1], shown in Figure 1. There are N long-lived TCP flows sharing a congested link with capacity C. Each TCP flow has a different round-trip-time (RTT) and the average (harmonic average) RTT of flows is $\overline{\mathrm{RTT}}$. As a result of the TCP congestion avoidance algorithm, the queue size $Q(t)$ and the sum of the window sizes follow sawtooth patterns. Here, we use the same queue management scheme as in [1] – the commonly deployed drop-tail algorithm at router $R1$: when $Q(t)$ becomes equal to buffer size B, newly arrived packets are dropped.

Our goal is to analyze the collective behavior of the different TCP flows, especially how they scale for large N. In order to reproduce the results in [1], this setup uses only long-lived TCP flows sharing a common congested link. Previous research has shown that other types of flows or congested links elsewhere in the network do not play a significant role in buffer sizing [1, 8]. We extended the study by considering more general cases including large round trip time variations, different percentages of short flows, different window size caps, and longer simulation time, etc. In all cases considered, we found the same qualitative behavior on fairness versus synchronization. Thus, for this paper, we will focus on this simple yet critical setup to get a clear understanding of the underlying mechanisms. Explorations of more complicated systems can be expanded from this base study.

The results from [1] are confirmed by our *ns2* simulations as shown in Figure 2 where the variations in the queue length (Figure 2(a)) and the window size (Figure 2(b)) decrease with increasing number of flows N. Figure 2 shows the results using three different values of N: 100, 300, and 1000. The capacity of the bottleneck link scales with the number of flows [3], N*1Mb/s, and the round trip time varies between 80ms to 100ms among the flows. The size of each packet is 960 bytes and the duration of the whole experiment is 50 seconds. To further study packet dropping, Figure 2(c) shows that when the buffer becomes full and packets are dropped at the intermediate router, there are roughly N packets being dropped during each congestion period. This result offers a key clue to understanding the interplay between fairness in packet dropping and synchronization among the flows, as will be discussed in Section 2.2. The origin of the N dropped packets per congestion period is explained in the Appendix.

To analyze the fairness among the flows, for each flow, we take the average of congestion window sizes over time, and plot the histogram in Figure 3(a) for the case with N=1000, which is representative of the other simulations. The interesting observation here is the extremely long tail of the histogram, which indicates that there are flows whose average congestion window size grows well beyond that of the majority of the flows. In TCP-Reno, changes in congestion window size happen as a direct consequence of packet drops. Therefore, these observations suggest that packet drops might not be fairly distributed amongst flows.

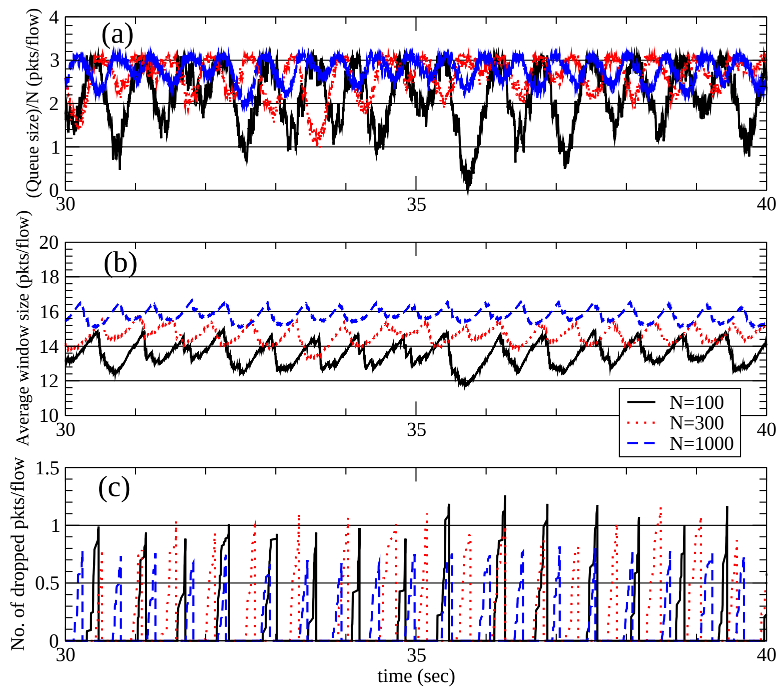

Fig. 2. Scaling with the number of flows from *ns2* simulations using the drop-tail algorithm. **a)** Average queue length, **b)** average window size, and **c)** average number of packets dropped over each congestion cycle. The averaging is done over the number of flows N. Three different simulation results are shown with N=100, 300, and 1000. Other parameters are kept the same for the three simulations: $(C * \overline{\mathrm{RTT}})/N = 12$ packets/flow, $B/N = 3$ packets/flow, $\mathrm{RTT} = [80, 100]$ms.

To further study fairness in packet drops, we make the following definition.

Definition 1 *In the setting defined above, let us denote the number of packet drops as seen by flow i between times t_1 and t_2 as $D(i, t_1, t_2)$. We denote the vector of all drops between t_1 and t_2 as $\boldsymbol{D}(t_1, t_2)$. Unfairness in packet drops between t_1 and t_2, denoted by $U(t_1, t_2)$, is defined as the variance of $\boldsymbol{D}(t_1, t_2)$. Fairness is defined as:*

$$F(t_1, t_2) = \frac{1}{1 + U(t_1, t_2)}. \tag{1}$$

Based on this definition, fairness is a number greater than zero and less than or equal to one. Maximum fairness is when all flows see the same number of drops. In this case, unfairness, or the variance in packet drops, is zero, which makes fairness equal to one. By default we set $t_2 = t_1 + \overline{\mathrm{RTT}}$.

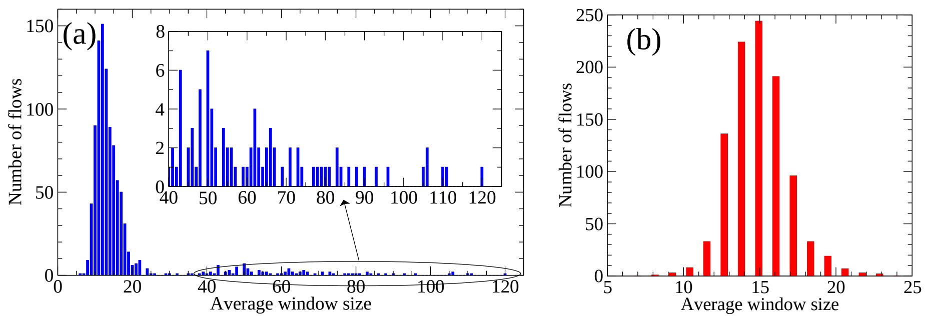

Fig. 3. Histogram of time-averaged window size of individual flows from *ns2* simulations based on (a) the drop-tail algorithm and (b) the randomized drop-tail algorithm. These results are obtained from the same configurations as in Figures 2 and 5 for N=1000.

Figure 4 shows the cumulative distribution function (CDF) of fairness. With drop-tail queuing, for about 55% of the time, fairness is equal to 1. These are times when no packets are dropped, and thus the system is completely fair. However, when there are packet drops in the system (the remaining 45% of the time) the system is very unfair in dropping packets; fairness is less than 0.5 for more than 30% of the time. This graph suggests that the combination of TCP-Reno and the drop-tail queue management scheme is not fair in dropping packets. The other line in this figure will be discussed later in Section 2.2.

Unfairness in packet drops is not surprising. TCP-Reno injects packets to the system in a bursty manner [2]. Whenever such a burst of packets arrives to a full or nearly full queue in an intermediate router which uses drop-tail, a large number of those packets are expected to be dropped quite unfairly since other flows in the system see almost no drops. A fair packet dropping scheme on the other hand, would drop packets from all flows in the queue rather than the flow whose burst has reached the tail of the queue. Now, if by any chance a flow's bursts arrive at times when the queue is not nearly full for a number of RTTs, that specific flow will not see any packet drops, thus its congestion window size will grow significantly compared to the rest of the flows.

2.2 Fairness leads to synchronization

If desynchronization among TCP-Reno flows comes from unfairness in packet drops, should increasing fairness lead to synchronization? In this section, we will try to find out whether this is the case or not.

One way to increase fairness in packet drops is to add randomness. We start with a simple experiment by introducing some randomness while not deviating too much from the drop-tail algorithm. We call this the randomized drop-tail scheme: when the queue length is less than 90% of buffer size, none of the packets

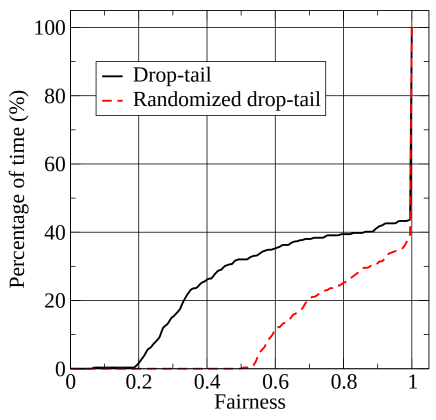

Fig. 4. Comparison of cumulative distribution functions of fairness over time between the drop-tail and the randomized drop-tail schemes.

are dropped; otherwise, each packet is dropped according to a probability that is calculated as follows:

$$p_{\text{drop}} = 10 \times \left(\frac{\text{Queue Length}}{\text{Buffer Size}} - 0.9 \right) . \tag{2}$$

In other words, the drop probability increases linearly as a function of queue length from 90% to 100% of the buffer size. Using this algorithm, packets are dropped only when the buffer is nearly full, which therefore does not deviate much from the drop-tail algorithm. Note that this randomized drop-tail scheme is different from the well-known "Random Early Detection" (RED) queue management scheme [23]. The latter is a much more sophisticated scheme where the packet dropping probability depends not only on the queue length but also on the history of the queue. RED starts to drop packets even when the queue length is short, much earlier than the drop-tail scheme.

Figure 5 shows the results from this randomized drop-tail algorithm with otherwise the same configuration as in Figure 2. With randomness introduced, the queue size variation and the total window size variation become larger than in Figure 2. Furthermore, these variations do not decrease any more with increasing N. Both the queue length (Figure 5(a)) and the window size (Figure 5(b)) enter into periodic sawtooth behavior, indicative of global synchronization. Interestingly, the number of packets dropped over each congestion cycle (Figure 5(c)) still comes up to be close to N, similar to the previous case for the unfair packet dropping (Figure 2(c)).

The fairness in this scenario is shown in Figure 4. The CDF of fairness indicates that adding randomness in packet dropping increases fairness: for about 60% of the time fairness is equal to one (when there are no dropped packets), and for the rest of the time (when there are packet drops) it is above 0.5, higher than in the drop-tail scenario. The fact that the randomized drop-tail recovers fairness in the flows is further illustrated in Figure 3(b), where the time-averaged window sizes of the different flows fall into a rather narrow distribution about the mean.

The results obtained from the randomized drop-tail simulations can be understood quantitatively by a mean-field method [22]. In the specific case when the N long flows have identical RTTs, there should be N packets dropped during each congestion cycle. When these N dropped packets are spread randomly among the N flows, about 63% of the flows see packet dropping from probability calculations [22]. Therefore, the mean-field calculation predicts global synchronization for large N when packets are dropped fairly within the drop-tail scheme. Fair packet dropping means that the packet dropping probability for each flow is proportional to the flow size, i.e., its current congestion window size.

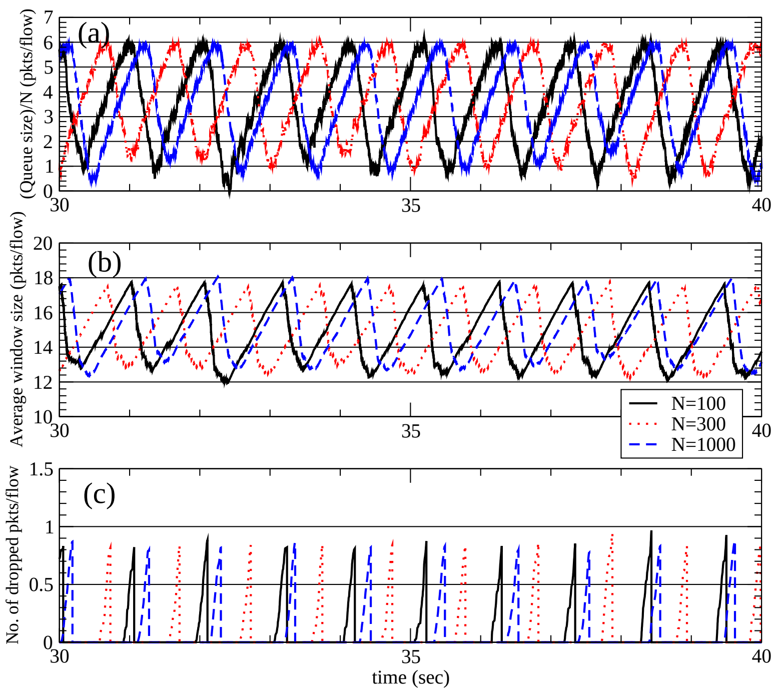

Fig. 5. Scaling with the number of flows from *ns2* simulations using the randomized drop-tail algorithm. **a)** Average queue length, **b)** average window size, and **c)** average number of packets dropped over each congestion cycle. The configurations are identical to those in Figure 2 except for $B/N = 6$ packets/flow to ensure the full utilization of the link.

To study whether the conclusions drawn above are robust against different parameter variations, we carried out *ns2* simulations with many different conditions where we varied RTTs and their variations, uplink capacity, buffer size, number of flows, with and without congestion window size limits, and the mixing of short flows. In all cases considered, we observed the same behavior: the drop-tail algorithm gives unfair bandwidth partitioning, whereas the randomized drop-tail is much more fair. The results from a particular set of simulations with the randomized drop-tail are shown in Table 1, where the RTTs and the number of flows are varied. The overall behavior remains the same: both the total window size variation and the percentage of flows with dropped packets do not decrease as N increases. For the cases where the round trip times vary little (from 89ms to 91ms), the results match very well with the mean-field analysis results - there are about 63% of flows with dropped packets [22]. This is consistent with and explains the simulation result reported earlier that there are over 60% flows with packet drops during each congestion cycle [8].

Figure 6 is a summary of the difference between the drop-tail and the randomized drop-tail algorithms. When the number of flows N increases, the percentage of flows that have dropped packets per congestion cycle decreases in the drop-tail scheme, in contrast to the randomized drop-tail scheme where the percentage of dropped flows does not change appreciably with N. The results from the above *ns2* simulations show that once fairness is recovered in the drop-tail queue management scheme, as in the randomized drop-tail scheme, the simulation results match very well to the theoretical calculations based on the mean-field analysis [22]. However, this fairness in packet dropping causes synchronization among flows, thus requires larger buffer sizes.

RTT	N	$\langle W_{\max}\rangle$	$\langle W_{\min}\rangle$	$1-\frac{\langle W_{\min}\rangle}{\langle W_{\max}\rangle}$	$\frac{N_{\text{drop}}}{\text{flow}}$	Flows w/ dropped packets
(ms)		(pkts)	(pkts)	(%)	(pkts)	(%)
[89	100	17.6	11.5	34.5	0.88	64
-	300	17.7	11.8	33.6	0.86	62
91]	1000	17.7	11.8	33.4	0.86	62
[80	100	17.4	12.5	28.5	0.87	52
-	300	17.5	12.5	28.7	0.85	53
100]	1000	17.9	12.8	28.4	0.83	53
[60	100	17.3	13.4	22.2	0.79	42
-	300	18.4	13.4	27.4	0.77	45
120]	1000	17.4	13.2	24.4	0.76	47

Table 1. *ns2* simulations with the randomized drop-tail algorithm by varying the range of round trip time and the number of long flows. The other parameters are the same as in Figure 5.

The fact that fairness causes synchronization can explain the difference between the results observed by Appenzeller *et al.* [1] and those of Raina and Wischik [7]. Appenzeller *et al.* do not see any synchronization in the system, since they use TCP-Reno with drop-tail, which is unfair in packet drops. On the other hand, Raina and Wischik base their analysis on a fluid model, which

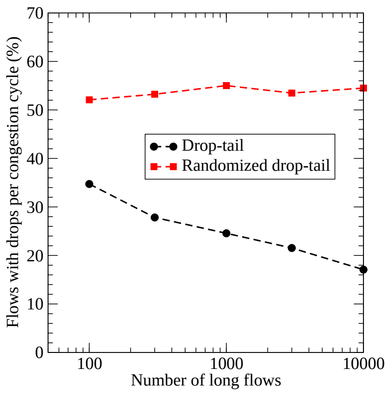

Fig. 6. The percentage of flows with dropped packets per congestion cycle as a function of the number of long TCP flows: the comparison between the drop-tail and the randomized drop-tail algorithms. The configurations are identical to those in Figure 2 with $\text{RTT} = [80, 100]\text{ms}$.

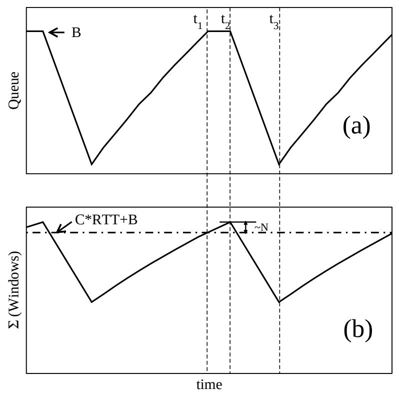

Fig. 7. Sawtooth pattern of both the (a) queue size and (b) sum of windows of all flows. At t_1, the buffer becomes full and packets start to be dropped. This continues for one round-trip time until t_2 when some flows start to half their windows and pause sending packets. At t_3, all flows with dropped packets have reduced their windows and the sum of windows reaches minimum. Between t_3 and $t_1 + T$, all the flows resume sending packets and the buffer slowly fills up.

assumes fairness in packet dropping. As a result they predict synchronization among flows, and an unstable system.

3 Conclusion

In this paper, we investigated the relationship between fairness and desynchronization among TCP flows. We found that : 1) the desynchronization in the presence of a large number of long flows from the drop-tail scheme is a result of unfairness in packet dropping; and 2) if fairness is imposed, the drop-tail scheme leads to global synchronization of the TCP flows. This results in a periodic sawtooth behavior in the queue length and the total window size, during which the majority of the flows have dropped packets within each congestion cycle.

The studies in this paper provides insights on the effects of fairness in buffer sizing. One of the key factors that leads to the global synchronization is the fact that the packet drops occur during a short time period. One can avoid this by spreading out the packet drops over a longer period of time. Thus, it is feasible to achieve both fairness and desynchronization with full link utilization. Further research in finding the right scheme would be interesting and very much needed.

4 Acknowledgments

The authors would like to thank Professor Ashish Goel, Dr. Larry Dunn, and Damon Mosk-Aoyama for helpful discussions and feedback; and Dr. Guido Appenzeller for the scripts to reproduce results in [1].

References

1. G. Appenzeller, I. Keslassy and N. McKeown , “Sizing Router Buffers”, ACM SIGCOMM 2004.
2. M. Enachescu, Y. Ganjali, A. Goel, N. McKeown, T. Roughgarden, “Routers with very small buffers”, Proceedings of INFOCOM, 2006.
3. D. Wischik, N. McKeown, “Part I: Buffer sizes for core routers”, ACM SIGCOMM Computer Communication Review, July 2005.
4. N. Beheshti, Y. Ganjali, R. Rajaduray, D. Blumenthal, N. McKeown, “Buffer sizing in all-optical packet switches”, Proceedings of OFC/NFOEC, March 2006.
5. R. Shorten, D. Leith, “On queue provisioning, network efficiency and the transmission control protocol”, accepted to appear in IEEE/ACM Transactions on Networking, Dec. 2007.
6. G. Raina, D. Towsley, D. Wischik, “Part II: Control theory for buffer sizing”, ACM SIGCOMM Computer Communication Review, July 2005.
7. G. Raina, D. Wischik, “Buffer sizes for large multiplexers: TCP queueing theory and instability analysis”, EuroNGI, April 2005.
8. A. Dhamdhere, H. Jiang, C. Dovrolis, “Buffer sizing for congested Internet links”, Proceedings of INFOCOM, 2005.
9. A. Dhamdhere, C. Dovrolis, ”Open issues in router buffer sizing”, ACM SIGCOMM Computer Communications Review, Jan. 2006.
10. D. Wischik, “Buffer requriements for high-speed routers”, ECOC 2005.
11. D. Wischik, ”Fairness, QoS, and buffer sizing”, ACM SIGCOMM Computer Communications Review, Jan. 2006.
12. J. Sommers, P. Barford, A. Greenberg, W. Willinger, “A SLA perspective on the router buffer sizing problem”, University of Wisconsin Techical Report, 2006.
13. G. Appenzeller, N. McKeown, J. Sommers, P. Barford, “Recent Results on Sizing Router Buffers”, Proceedings of the Network Systems Design Conference, Oct. 2004.
14. S. Gorinsky, A. Kantawala, J. Turner, “Link buffer sizing: a new look at the old problem”, Proceedings of the IEEE Symposium on Computers and Communications, June 2005.
15. A. Lakshmikantha, R. Srikant, C. Beck, “Are small buffers feasible in high speed routers?”, in submission, 2006.
16. K. Huang, “Statistical Mechanics”, 2nd edition, Wiley, 1987.
17. network simulator 2.
18. D. Comer, “Internetworking with TCP/IP”, Vol 1, 5th edition, Prentice Hall, 2005.
19. F. Baccelli, A. Chaintreau, D. Vleeschauwer, D. McDonald, “A mean-field analysis of short lived interacting TCP flows”, SIGMETRICS, 2004.
20. M. Marsan, M. Garetto, P. Giaccone, E. Leonardi, E. Schiattarella, A. Tarello, “Using partial differential equations to model TCP mice and elephants in large IP networks”, Proceedings of INFOCOM, 2004.
21. M. Wang, X. Zou, F. Bonomi, B. Prabhakar, “Elephant traps using a random sampling algorithm”, in preparation.

22. M. Wang, Y. Ganjali, "Unifying buffer sizing results through fairness", Technical Report, HR06-HPNG-060606, Stanford University, June 2006. http://yuba.stanford.edu/techreports/TR06-HPNG-060606.pdf
23. S. Floyd and V. Jacobson, "Random Early Detection gateways for congestion avoidance", IEEE Transactions on Networking, August 1993.
24. R. K. Pathria, "Statistical Mechanics", 2nd edition, Butterworth-Heinemann, 1996.

APPENDIX: Mean-field Analysis

In this appendix, we explain the origin of the claim that there are N packets dropped in each congestion period, where N is the total number of long-lived TCP flows sharing the congested link.

The network configuration considered here is shown in Figure 1. As observed in typical $ns2$ simulations, both the queue size and the total window size follow a sawtooth pattern over time. We study the sawtooth pattern that has stabilized to a periodic function with period T, which serves as a good representative.

We dissect the sawtooth pattern into different segments, as illustrated in Figure 7. There is a period of time when the buffer is full and packets are being dropped. During this period, (between t_1 and t_2 in Figure 7) the window size keeps increasing until some flows detect the dropped packets. These flows in turn reduce window sizes and temporarily stop sending packets. During this time, between t_2 and t_3, the total window size declines and the buffer starts to deplete. When the flows that have dropped packets receive enough acknowledgements, they resume sending packets. During this segment, between t_3 and $t_1 + T$, the total window size increases and the queue length slowly builds up.

When the buffer first becomes full, denoted by time t_1, router $R1$ starts to drop packets. At t_1, the sum of the windows of all the flows, $W(t_1)$, consists of packets in transit $C \times \overline{\mathrm{RTT}}$ plus those that are queued in the buffer B. Router $R1$ continues to drop packets while the buffer remains full until some of the flows detect the dropped packets and reduce their window sizes. The feedback for flows to reduce window sizes comes after the un-dropped packets complete a round-trip, starting at $t_2 = t_1 + \overline{\mathrm{RTT}} + B/C$. The extra packets sent during the time period between t_1 and t_2, i.e., after the buffer is full and before the senders receive the feedback to reduce the window sizes, are dropped. Each flow increases its window by 1 for each round-trip before it receives the dropped packets feedback. The time duration from t_1 to t_2 is roughly one round-trip time. There are total N flows. Thus, the total number of packets injected into the network at t_2 is $W(t_2) = C \times \overline{\mathrm{RTT}} + B + N$. Therefore, the number of packets dropped during t_1 to t_2 is

$$W(t_2) - W(t_1) = C \times \overline{\mathrm{RTT}} + B + N - (C \times \overline{\mathrm{RTT}} + B) = N, \tag{3}$$

After t_2, some senders start reducing their window sizes, the queue is not full any more, and no packets are dropped either until the next cycle at $T + t_1$. Thus, the total number of packets dropped within one congestion cycle is N.