Head-to-Tail: Managing Network Load through Random Delay Increase

Stylianos Dimitriou, Vassilis Tsaoussidis
Electrical and Computer Engineering Department, Democritus University of Greece
{sdimitr, vtsaousi}@ee.duth.gr

Abstract

Window-based congestion control is typically based on exhausting bandwidth capacity, which occasionally leads to transient congestion. Moreover, flow synchronization may deteriorate conditions further, leading to persistent or more severe congestion, which is experienced by flows through increasing queuing delays and packet retransmission. Head-to-Tail is a new approach to queue scheduling that aspires to alleviate this problem. When conditions at the router's buffer indicate high risk for congestion, Head-to-Tail delays packets intentionally to fabricate the senders' impression about the network load. This implicit signal to reduce the transmission rate allows for a responsive behavior prior to congestion. In this paper, we evaluated Head-to-Tail with TCP Vegas and compared it with RED and other TCP variants. The initial results indicate that congestion events and retransmissions can be significantly eliminated.

1. Introduction

The Transmission Control Protocol has powerful mechanisms for detecting congestion and for recovering after congestion happens. Recovery typically depends on the characteristics of congestion itself: severe congestion, which is detected by timeout triggers a radical transmission rate decrease, while transient congestion, which is detected by 3 DACKs, triggers a milder reaction. Several TCP versions employ also congestion avoidance mechanisms. Such mechanisms monitor the network load by measuring the Round Trip Time (RTT) per window, the inter-packet gap, the buffer occupancy or a combination of the above. This mechanism of detecting network conditions allows for a new mechanism to implicitly report network over-utilization. That is, increasing the queuing delay of selected packets intentionally will cause an increase of the measured load at the corresponding senders, which could trigger a reduction of the transmission rate. Clearly, the intentional delay increase seems to collaborate well with the transport protocols that do employ measurement-based congestion avoidance. Additionally, the mechanism has the potential to work well also with congestion control instead of avoidance, since the RTT measurements are central to all protocol variations. For example, in standard TCP Reno the RTT sets the timeout values, which in turn, determines the scheduling of retransmitted packets. Furthermore, the RTT determines the pace of received acks, which in turn determine the pace of packet transmission.

The proposal to increase delay intentionally is associated with two main issues.

1. How much and in what occasions shall we increase packet delay in the queue? The increase should be significant enough in order to be measured; and small enough to avoid a spurious timeout at the sender. If the granularity of measurement is not appropriate, the increase will go wasted.
2. Delay increase cannot apply to all queued packets – an overflow in that case cannot be avoided. Since the mechanism will apply to selected packets only, a set of packets will be favored at the expense of others. What is the impact of such policy on fairness?

Departing from the two aforementioned issues, we investigate the desired properties of a Rearrange Probability Function. In section 2 we provide the necessary background and discuss the related work. In section 3 we detail the Head-to-Tail algorithm and in section 4 we determine the experimental scenarios. In section 5 we evaluate the performance of various Rearrange Probability Functions using TCP Vegas as a representative measurement-based protocol. We extend our evaluation further in section 6, with more TCP versions and Active Queue Management schemes. In section 7 we conclude with some main remarks.

2. Background and Related Work

There are two main policies at the router that impact flow performance: (i) the dropping policy and (ii) the scheduling policy. Scheduling typically manages priorities of packets. Dropping is focused on penalizing high-bandwidth-consuming flows. In [3] Floyd and Fall identify high-bandwidth flows by using RED's [5] drop-history. The RED-PM (Preferential Dropping) algorithm [8] uses per-flow preferential dropping mechanisms. Per-flow preferential dropping with FIFO scheduling also use Core-Stateless Fair Queuing (CSFQ) [13] and Flow Random Early Detection (FRED) [7]. CSFQ marks packets with an estimate of their current sending rate. The router uses this information in conjunction with the flow's fair share estimation in order to decide whether a packet needs to be dropped. FRED maintains a state only for the flows which have packets in the queue. More packets buffered equals to increased dropping probability.

The CHOKe mechanism [12] matches every incoming packet against a random packet in the queue. If they belong to the same flow, both packets are dropped. Otherwise, the incoming packet is admitted with a certain probability. The Stochastic Fair Blue (SFB) [2] uses multiple levels of hashing in order to identify high-bandwidth flows and ERUF [11] uses source quench to have undeliverable packets dropped at the edge routers. On the other hand, SRED [10] caches the recent flows in order to determine the high-bandwidth flows.

Although HtT cannot be classified as a clear dropping or scheduling scheme, it has a common property with the aforementioned mechanisms, which is to implicitly indicate congestion status. In this context, HtT can be viewed as a new mechanism for Active Queue Management.

Prior to analyzing HtT, we highlight some major differences of various TCP variations for controlling congestion. We classify TCP into two major categories: (i) the AIMD-based standard versions such as Reno [6], New Reno [4] and SACK [9] and (ii) the measurement-based approaches such as Vegas [1], Real [15] and Westwood [14]. The first category mainly assumes that network is a black box. Each packet loss triggers homogeneous responses from all senders, which adjust their transmission windows multiplicatively at a fixed rate β. The increase rate is pre-determined by parameter α. Most measurement-based protocols, instead, complement the AIMD-based control with adjustments of the transmission window that correspond to detected network conditions. Such conditions are detected by measuring the RTT, the inter-packet gap, the packet loss rate, etc.

It is important to highlight a main difference in the design of the two approaches. The first category is responsive to packet losses, while the second category is also responsive to other events as well, such as, for example, the detected network load. Of course, the first category is also somewhat responsive to RTT measurements, since those affect the timeout. However, a single RTT increase will not trigger in that case any transmission window reduction but it will cause a small rate reduction due to further delays of acks.

3. Head-to-Tail description and justification

Departing from the aforementioned observation, we design HtT to cause intentional delays and hence indicate implicitly to the corresponding senders the urgency to reduce their rate. The indication causes clearly different reactions at the senders, depending on their congestion avoidance and control mechanism; however, all types of mechanisms are more or less reactive to the additional delay.

The proposed scheme can work well with standard scheduling schemes such as FIFO and also with standard dropping schemes, such as DropTail. After the arrival of a new packet, the router decides whether or not to change the existing order (rearrange) of some packets based on two criteria: (i) the current queue length and (ii) the selected Rearrange Probability Function (RPF). Note that the RPF determines the probability that corresponds to some particular length of the queue.

When a new packet arrives, the router generates a pseudo-random number between 0 and 1. If it is smaller than the probability that corresponds to the queue's length, the queue shifts backwards the position of selected packets. Those packets cannot be randomly chosen, but instead they have to be selected in order to maximize the delay impact and increase the probability of informing the sender. That said, packets are moved from Head to Tail. However, our policy is not biased: the time and probability of selection is associated with the queue length, the random number generation and the RPF. In this context, randomness is also associated with the packets selected and hence our design is not expected to degrade system fairness. Also note that the diversity of flows that are implicitly informed about the network conditions is increased in proportion to flow contention.

By and large, the importance of the Rearrange Probability Function becomes obvious. RPF depicts the probability for the rearrange to occur at each corresponding length of the queue. It may be a constant function, a pulse, a first grade polynomial, or it may be composed by more than one functions of any type (see Figure 1).

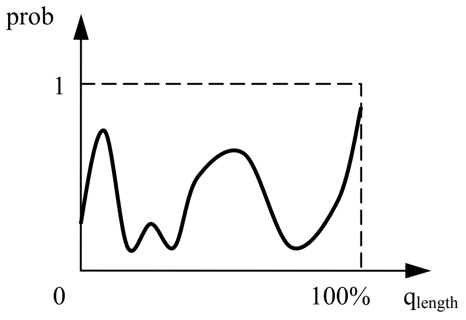

Figure 1. Example of a Rearrange Probability Function

On the x-axis we depict the length of the queue where 0% is an empty queue and 100% is a full queue. On the y-axis we depict the rearrange probability. The maximum probability is 1, meaning that each time the router reaches this length, it rearranges some packets. Every position in the queue has its own probability; however we avoid rearranging for packets at both empty and full queues. Empty queues need no supportive mechanisms while full queues may be handled better with drops. Hence, our approach intends to mainly regulate traffic during the normal operational phases of the queue – not the extreme ones.

4. Determining RPF Shape

Our initial target was to determine an appropriate RPF using TCP Vegas flows. We made five sets of simulations with five different types of RPF: constant functions, pulse functions, two first and two fourth grade polynomials convex with one peak and two fourth grade polynomials concave. We can see indicative forms of the functions in Figure 2.
For the (3), (4) and (5) RPF we chose to have one peak. This was deemed appropriate but also convenient since otherwise, we had to search for twice more peaks and because we decided that maximum probability should correspond to a single point. Variables *yv*, *xv*, *xv1*, *xv2* by definition take values from 0 to 1. During the simulations and in order to control the computational time, we used a 0.1increase step for the above variables. Simulations were done with ns-2. We used a Dumbbell topology with two routers, N senders and N receivers, as we see in Figure 3.

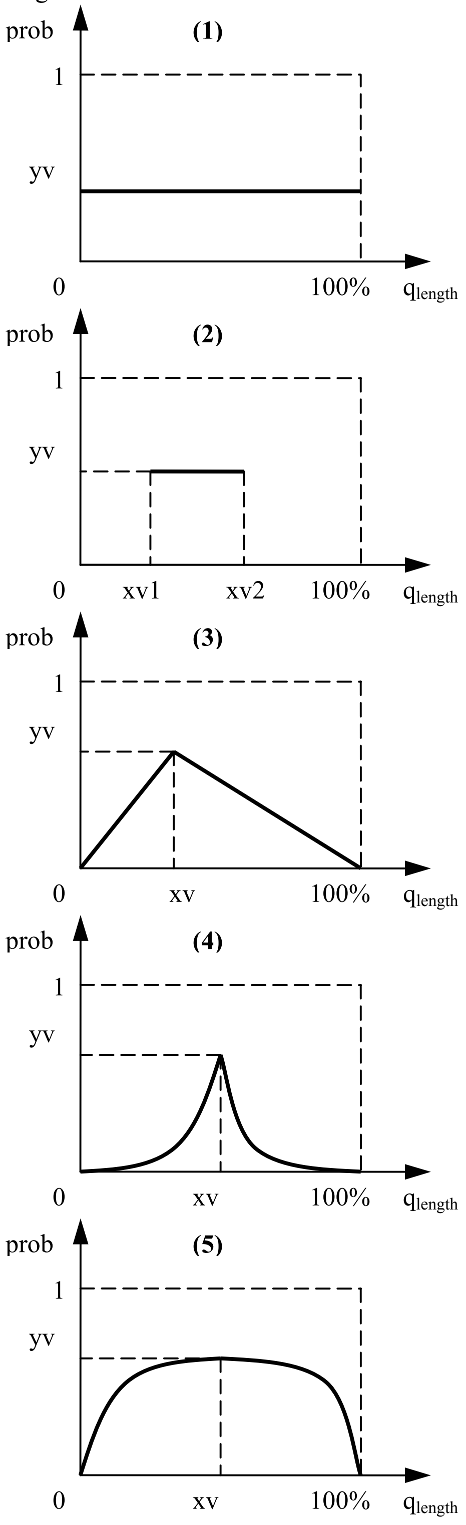

Figure 2. The 5 types of RPF that we examined during our simulations

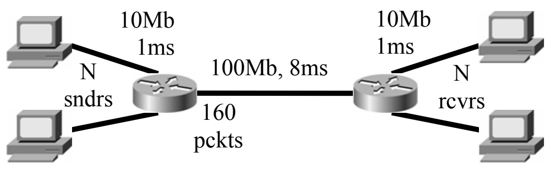

Figure 3. The Dumbbell topology used

Every simulation's duration was 70 seconds. Every 10 seconds the number of flows changes taking values from 10 to 500, representing both the dynamics of contention increase and contention decrease. The first time we simulated the scenario without HtT and in the following phase we tried all 5 RPFs with all the possible values for their variables (step 0.1) by setting the number of packets to rearrange *pv*=6. Note that this number itself can be a subject of research and optimization; however, presently, we have used a fixed number of packets.

Since a comparative advantage of HtT is the reduction of packet drops, we mainly focused on reducing retransmissions even at the same levels of Goodput, Throughput and System Fairness. Hence, retransmission overhead was a major performance evaluation metric.

We also considered very important to show that the proposed scheme does not harm existing AIMD-based versions in favor of measurement-based flows.

5. On the Choice of the RPF

In order to examine thoroughly the form of the RPF we simulated the above scenario for every possible value of the variables, with step 0.1 (apart from the first, where we used smaller steps). This way, we performed a total of about 900 simulations: 30 of them were for the constant RPF, 550 of them were for the pulse-shaped RPF and for each of the (3), (4) and (5) RPF we performed 110 simulations. The results of each RPF are depicted in increasing order. The y-axis depicts the number of the retransmitted packets. Without HtT, the number of RetPacks was 66320. This can be seen on the graphs with the transparent circle. The black circle in Figure 10 is the point which we choose as the best RPF and possibly will be used for further implementation and evaluation of the algorithm. For the third and fourth RPF, we give diagrams depicting the areas where we have improvement of the number of Retransmitted Packets (RetPacks). Since we had three variables for the second RPF, we couldn't display a similar diagram. Area diagrams for the first and last RPF weren't necessary as we couldn't achieve any amelioration. The blue (light gray) areas indicate the points where the total number of the retransmitted packets is reduced.

5.1. Constant and Fourth Grade Concave RPFs

As we can see in Figures 4 and 5, all the simulations produced worse results than the original algorithm. Even when we used 0.001 constant rearrange probability, packet losses were slightly greater.

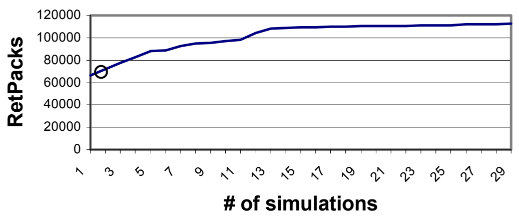

Figure 4. Constant RPF

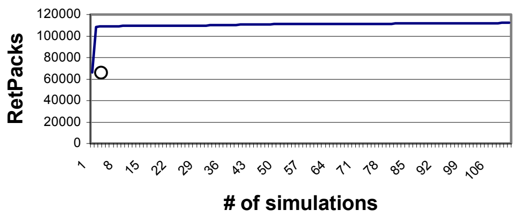

Figure 5. Fourth Grade concave RPF

The reason for this behavior is that rearranging packets near the end of the queue, at least doubles their queuing delay. This raises the total delay of the packet, exceeding the factor 2·RTT (which equals usually to the RTO) in fast links, and causes the sender to consider the packet lost and retransmit it.

We should notice that all the results for the Fourth Grade concave RPF were almost identical, around 11000 packets. Because of the form of this RPF, altering the peak point among neighboring points will not alter the probabilities in different percentages of the queue length. This leads to the conclusion, that the value of probability isn't very important when we examine the rearrange probabilities for each queue's length. Great fluctuations of the results are observed when probability approaches 0 and departs from 0.

5.2. First Grade and Fourth Grade convex RPFs

For those two RPFs, 36% and 73% of their peaks gave lower RetPacks than 66320 packets. What we can see in Figures 6 and 8 is that the results are being divided in two clusters, leading to the conclusion that some values of *yv* and *xv* can lead to an extraordinary number of congestion events and retransmissions. This calls for self-adaptive systems that make judgments, based on measurements, in case a specific Rearrange Probability Function mitigates the network's load, or causes unnecessary delays.

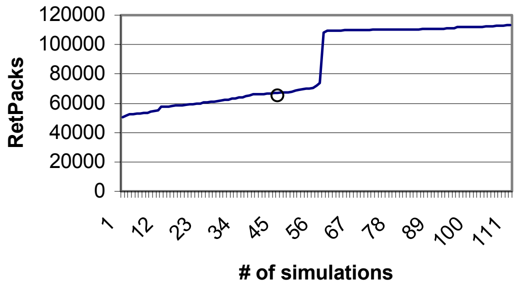

Figure 6. First Grade RPF

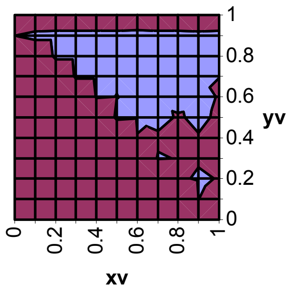

Figure 7. Area Graph for First Grade RPF

Low performance of the First Grade RPF renders it practically useless. Instead, the Fourth Grade convex RPF had the highest performance among all RPFs evaluated. In Figure 9, we can see the areas where Fourth Grade convex RPF succeeds in producing better results, system-wise. First, it becomes clear, why someone should avoid rearranging packets near the head or near the tail of the queue. Second, the red (dark gray) and blue (light gray) areas are not mixed: the blue area rests in the middle of the graph and the red area occupies the top and the bottom of the graph. Picking a peak is easy, since any point with *yv* greater than 0.2 and smaller than 0.8 will work. This means that traffic regulation appears to be manageable with certain, well-known guidelines.

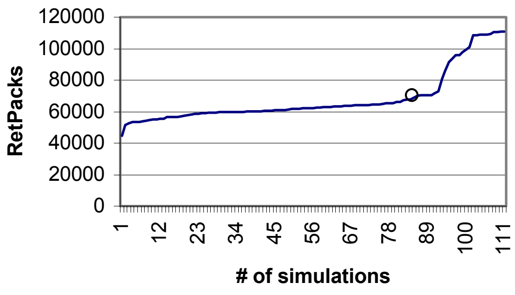

Figure 8. Fourth Grade convex RPF

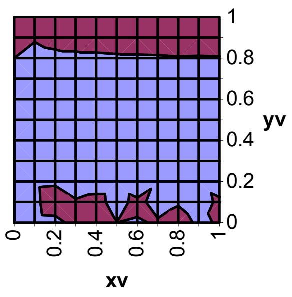

Figure 9. Area Graph for Fourth Grade convex RPF

5.3. Pulse RPF

Due to the nature of Pulse RPF, which is associated with a wide range of values, we cannot easily display results and proper conclusions. However, pulses with *xv2* equal to 1, generate congestion and packet losses; again, we exclude extreme cases of empty and full queue.

By and large, 66% of the simulations of Pulse RPF were positive, rendering the function comparable to the Fourth Grade convex RPF and hence, a good alternative. Because of the fact that Pulse is the only function that can have zero probability near the end of the queue, we can choose this function for calibrating Head-to-Tail further. Specifically, the point where *xv1*=0.1, *xv2*=0.9 and *yv*=0.1 and RetPacks=50927, presents over 23% reduction of retransmitted packets. This point can be seen on Figure 10 with the black circle. Pulse RPF has an advantage over Fourth Grade convex RPF since it does not require complicated calculations for each probability.

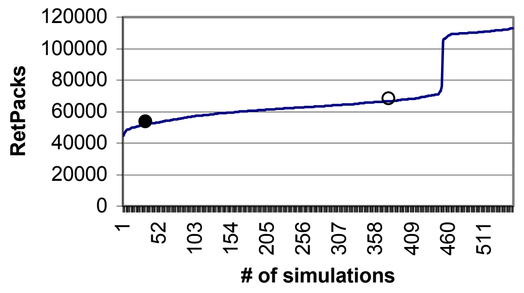

Figure 10. Pulse RPF

6. Examining different versions of TCP

Using the same topology as previously, we simulated Tahoe, Reno, NewReno, Westwood and Real flows, using RED, Droptail with FIFO, and Droptail with HtT. The results of these simulations are depicted in Figure 11.

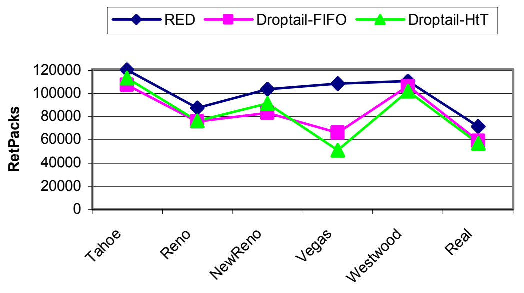

Figure 11. Different TCP versions simulations with RED, FIFO Droptail and HtT Droptail

HtT does not exhibit, comparatively, a behavior that harms Reno-like flows, more than RED and Droptail-FIFO. Instead, it appears clearly as the algorithm of choice, since in most cases it reduces further the amount of retransmitted packets. Note that all other metrics (e.g. goodput, fairness) do not exhibit any statistically significant difference. We do not present those figures here due to space limitations. Also note that, by altering the RPF (or its parameters), we can adjust the flows behavior further.

7. Conclusions and Future Work

We have shown that there are alternative ways to regulate queue traffic, without marking or dropping packets in the queue. HtT is an algorithm towards proactive congestion management and allows for designing sophisticated protocols that detect the network load. The information passed by HtT is, by and large, similar for all flows. However, different types of protocols interpret it differently. Hence, the impact on different versions of protocols is also different. Clearly, the granularity of measurements is also important. In this context, enhanced protocols and sophistication could promote the proposed scheme further.

8. References

[1] Brakmo et al, "TCP Vegas: New Techniques for Congestion Detection and Avoidance", SIGCOMM 94
[2] W.-C. Feng, K. G. Shin, D. Kandlur, and D. Saha, "Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness", Proceedings of the Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM-01), Volume 3, pp. 1520—1529. Los Alamitos, CA: IEEE Computer Society, 2001
[3] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the Internet", IEEE/ACM Transactions on Networking, 7(4):458-472, 1999
[4] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno modification to TCP's fast recovery algorithm", RFC 3782, April 2004
[5] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, 1(4):397-413, August 1993
[6] V. Jacobson, "Congestion Avoidance and Control", SIGCOMM Symposium on Communications Architectures and Protocols, pages 314–329, 1988
[7] D. Lin and R. Morris "Dynamics of random early detection", SIGCOMM '97, pages 127-137, Cannes, France, September 1997
[8] R. Mahajan, S. Floyd, and D. Wetherall, "Controlling high-bandwidth flows at the congested router", 2001
[9] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment and Options", RFC 2018, IETF, October 1996
[10] T. Ott, T. Lakshman and L. Wong, "SRED: Stabilized RED", IEEE Infocom 1999, New York, USA, March 1999
[11] A. Rangarajan and A. Acharya, "ERUF: Early Regulation of Unresponsive Best-Effort Traffic" Proceedings of ICNP'99, October 1999
[12] Rong Pan, Balaji Prabhakar, and Konstantinos Psounis, "CHOKe: a stateless AQM scheme for approximating fair bandwidth allocation", Proceedings of IEEE Infocom, March 2000
[13] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fair queuing: Achieving approximately fair bandwidth allocations in high speed networks", SIGCOMM, pages 118-130, 1998
[14] A. Zanella, G. Procissi, M. Gerla, M. Y. Sanadidi, "TCP Westwood: Analytic Model and Performance Evaluation", IEEE Globecom, November 2001
[15] C. Zhang, V. Tsaoussidis, "TCP-Real: Improving Real-time Capabilities of TCP over Heterogeneous Networks", NOSSDAV 2001