Hindawi Publishing Corporation
Research Letters in Communications
Volume 2008, Article ID 704878, 5 pages
doi:10.1155/2008/704878

Research Letter

A Simple Mechanism for Throttling High-Bandwidth Flows

Chia-Wei Chang and Bill Lin

Electrical and Computer Engineering Department, University of California, San Diego, La Jolla, CA 92093-0407, USA

Correspondence should be addressed to Chia-Wei Chang, chc019@ucsd.edu

Received 9 July 2008; Accepted 31 August 2008

Recommended by Jun Bi

This letter presents BREATHe, a simple packet dropping scheme for identifying and throttling unresponsive or misbehaving high-bandwidth flows during times of congestion. BREATHe is different from the existing active queue management techniques in that it uses heavy-hitter set analysis to identify highbandwidth flows rather than sampling or rate estimation. Specifically, BREATHe uses heavy-hitter set analysis to detect highbandwidth flows that exceed some target rate r_{limit} and preferentially drop packets from these flows. We show that the proposed mechanism is effective at throttling high-bandwidth flows using a small amount of state and low-complexity operation.

1. Introduction

The Internet provides a connectionless, best effort, end-to-end packet service that depends on the congestion avoidance mechanisms implemented at the end-points (like TCP) to provide good service under heavy load. However, bursty applications like VoIP and multimedia streaming can monopolize network bandwidth and cause rate-adaptive flows to suffer. As a result, these nonrate-adaptive UDP flows can aggressively use up increasingly more bandwidth as responsive flows back off in response to congestion, creating a vicious cycle of deteriorating TCP throughput. Therefore, it is necessary to protect responsive flows from unresponsive or aggressive flows and to provide a good quality of service (QoS) to all users.

Our work is motivated by the need for a simple stateless algorithm that can *protect rate-adaptive TCP friendly flows and approximate fair bandwidth allocation*. The basic idea behind BREATHe is that we block the faster flows until other flows can catch up. When a packet arrives at a congested router, it will be blocked if its flow ID (FID) is currently in the heavy-hitter set (HHS). Otherwise, it will be admitted into the queue with a probability depending on Qlen. The HHS is constructed using a small number of counters and the number of counters needed is relative to how much fairness we want to achieve. Note that BREATHe's packet dropping mechanism is only invoked when congestion occurs.

Simulation results show that BREATHe has higher protection ability than other AQM algorithms. BREATHe successfully provides not only protection ability, but also fairness. In our simulations, there are only 0.3% of TCP flows forced to stop. Most of TCP flows achieved throughput between 0.4 Kbps and 1.2 Kbps, versus the ideal fair rate of 0.97 Kbps.

This letter is organized as follows. Section 2 summarizes previous work in AQM. Section 3 describes the detailed BREATHe algorithm. Section 4 summarizes simulation results, and Section 5 provides concluding remarks.

2. Related Work

As discussed in [1], there are two broadly classified classes of router algorithms for achieving congestion control, "scheduling algorithms" and "queue management algorithms." The generic scheduling algorithm, exemplified by the well-known fair queueing (FQ) [2] algorithm, requires the buffer at each output of a router to be partitioned into separate queues and each queue will buffer the packets for one flow. However, this approach requires complicated per-flow state information, making it too expensive for wide deployment. Another notable scheme which aims to approximate FQ at a smaller implementation cost is stochastic fair queueing (SFQ) proposed by McKenney [3].

SFQ classifies packets into a smaller number of queues than FQ by using a hash function. Although it reduces FQ's design complexity, SFQ still requires around 1000 to 2000 queues in a typical router implementation to approach FQ's performance [4].

On the other hand, queue management algorithms have had simpler design problem from the outset. Given their simplicity, the hope is to approximate fairness with much lower hardware complexity. This class of algorithms is exemplified by random early detection (RED) [5]. It provides a solution to the global synchronization problems, which is caused by synchronized TCP backoffs. However, like Drop-Tail, RED is unable to penalize unresponsive flows [6] because every flow's packets suffer the same dropping probability during congestion periods. Consequently, misbehaving traffic can take up a large percentage of the link bandwidth and starve TCP flows. To improve RED's ability for distinguishing unresponsive users, a number of variants (like RED with penalty box and flow random early drop (FRED) [7]) have been proposed. However, these variants incur extra implementation overhead since they need to collect certain types of state information. Another method called stabilized RED (SRED) [8] aims to estimate the number of active connections and find candidates for misbehaving flows. It does this by maintaining a data structure, called the "Zombie list," which serves as a proxy for information about recently seen flows. Although SRED identifies misbehaving flows, it does not provide a simple router mechanism for penalizing misbehaving flows.

A notable scheme, CHOKe [9], is interesting in that it does not require any state information. The basic idea behind CHOKe is that the contents of the FIFO buffer provide "sufficient statistics" about the incoming traffic. Hence, packets belonging to a misbehaving flow arrive more frequently and are more likely to trigger comparisons and drops. As a result, a misbehaving flow has a higher dropping rate. But a recent study [10] shows that CHOKe only works for penalizing extremely heavy flows and cannot provide fairness to flows. Later, a differential dropping scheme called approximate fair dropping (AFD) [11] was proposed by the same authors to solve these problems. The idea behind AFD is to use rate information to decide the dropping probability. It uses a "shadow" buffer to record recent incoming traffic, and then estimates the rates of flows which are likely to be dropped by sampling. Although AFD is simpler than SFQ/FQ by using the same idea of rate shaping, it still needs large shadow buffers and long observation intervals to achieve fairness.

3. The BREATHe Algorithm

The design goal of BREATHe is to protect responsive TCP flows efficiently and approximate fair bandwidth allocation with a simple, stateless algorithm. BREATHe uses the heavy hitter data structure proposed in [12] to keep track of recent flows with many packets passing throughput the router. Note that CHOKe uses the current queue contents to decide dropping policy while AFD uses sampling and incoming rate estimation. Here, BREATHe uses the throughput to implement fairness. Based on our design, we can decide what QoS or how much fairness we want to achieve by adjusting the number of counters in HHS.

```
Require: $\psi$: Heavy hitter set.
Require: $C$: Total link capacity.
Require: $K$: Max number of counters in $\psi$ equal to $\frac{C}{r_{\text{limit}}}$.
Require: $P$: The admitted packet.
Require: $P_{\text{FID}}$: the flow ID of the admitted packet.
(1) initialize $\psi$ to empty;
(2) while $P$ do
(3)   if $P_{\text{FID}} \in \psi$ then
(4)     increment counter$[P_{\text{FID}}]$;
(5)   else if $|\psi| = K$ then
(6)     for all $i \in \psi$ do
(7)       decrement counter$[i]$;
(8)       if counter$[i] = 0$ then
(9)         remove $i$ from $\psi$;
(10)      end if
(11)    end for
(12)  else
(13)    insert $P_{\text{FID}}$ into $\psi$;
(14)    counter$[P_{\text{FID}}] = 1$;
(15)  end if
(16) end while
```

Algorithm 1: HHS ($\psi, r_{\text{limit}}, P$).

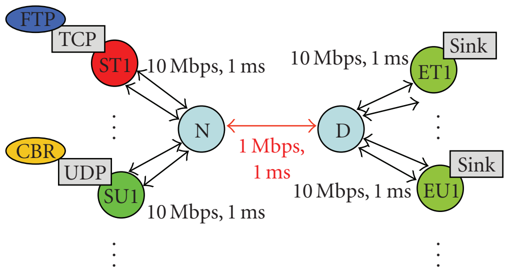
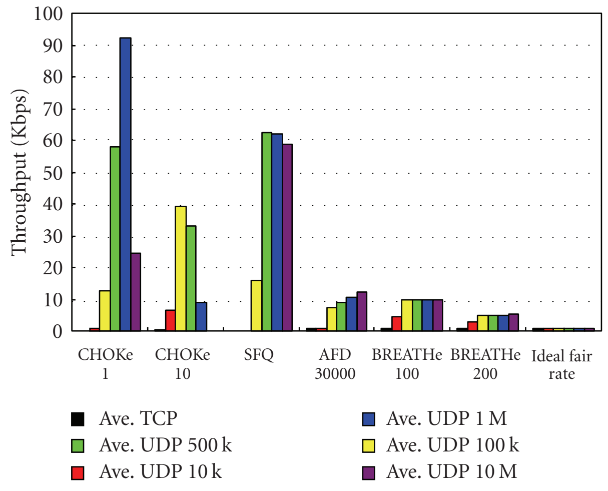
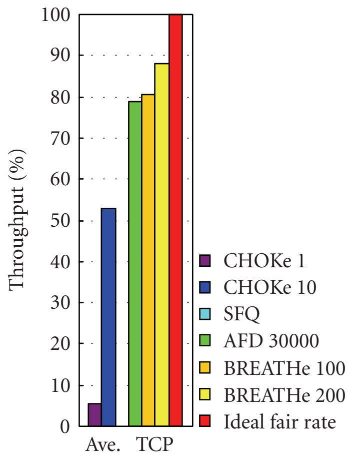


Figure 1: *Network configuration. m* TCP sources, *n* UDP sources.

BREATHe calculates the average occupancy of the FIFO buffer using an exponential moving average and also marks two thresholds, a minimum threshold $\min_{\text{th}}$ and a maximum threshold $\max_{\text{th}}$. If the average queue length (Qlen) is less than $\min_{\text{th}}$, every arriving packet is queued. If the average Qlen is greater than $\max_{\text{th}}$, every arriving packet is dropped. When the average Qlen is between $\min_{\text{th}}$ and $\max_{\text{th}}$, the arriving packet is dropped if its flow ID (FID) is in the HHS. Otherwise, it is dropped with a probability that depends on the Qlen to avoid the global synchronization and full-queue problems as RED. BREATHe does not require any special data structure (stateless algorithm). Compared to a pure FIFO queue, there are just a few simple extra operations that BREATHe needs to perform: build a heavy-hitter set, compare FIDs, and drop packets if it is in the set. Since BREATHe is embedded in RED, it inherits the good features of RED. The following is the details of HHS.

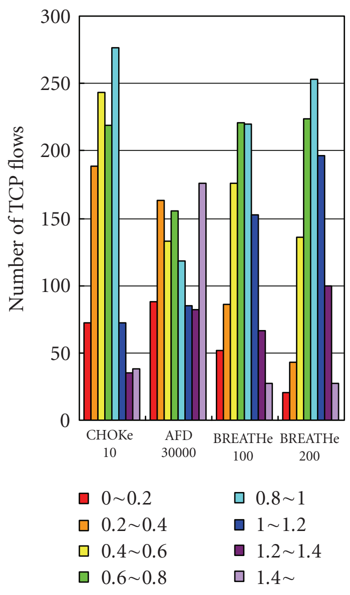


(a)

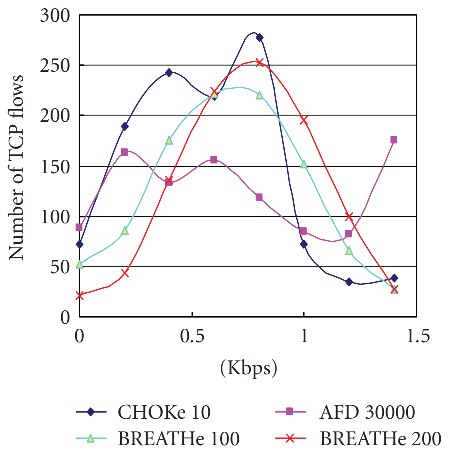


(b)

Figure 2: *Practical case.* There are 1000 TCP flows with 5 groups of UDP flows. The rate are 10 K, 100 K, 500 K, 1 M, 10 M, and the number are 10, 5, 5, 5, 5 separately. The fair rate for every flow is 1 M/1030 = 0.97 Kbps.



(a)



(b)

Figure 3: The ideal fair rate is 0.97 Kbps. We use the TCP rate distribution range to compare the fairness of these algorithms.

3.1. Heavy-Hitter Set

The main component of BREATHe is the heavy-hitter set (HHS). The goal of HHS is to record the majority flows which has been served by the router. As stated in [12] for finding frequent elements, if we want to find the majority flows which occupies at least θ of bandwidth, $\theta \in [0, 1]$, it only needs $1/\theta$ counters to do this job. The same applies when used in BREATHe: if the total link capacity is C, and we want to find flows with throughput more than r_{limit}, then we just need C/r_{limit} counters instead of implementing N (total number of flows) counters to catch them as shown in Algorithm 1. Initially, the HHS is empty. With the incoming packets, HHS will locate a free counter for it with its FID. If this FID is already in HHS, then the corresponding counter will be incremented by one. If all counters are already in use, and its FID is not in the HHS, then all counters are decremented by one and are garbage-collected if the corresponding value is equal to 0. The garbage collection function in Lines (6)–(11) can be *implicitly* implemented in constant time without actual decrements by using a slightly more sophisticated data structure. This constant time implementation of HHS is described in [12].

The idea is to use HHS information for dropping decisions. With the HHS, we could record the majority flows which have been served in this router. If there is no congestion, every packet can pass through the router, and the HHS does nothing to the packet dropping policy. If there is congestion, then the router will serve minority flows first and block the flows shown in the HHS. Effectively, the flows with heavy traffic will be blocked until other flows have comparable throughput. The spirit of this algorithm is to let small flows have a chance to “breathe” instead of dying.

4. Simulation Results

This section presents simulation results using the *ns2* simulator. The results are compared with some well-known AQM algorithms, including RED, CHOKe, SFQ, and AFD. A range of traffic mixes were used, including responsive TCP flows, small UDP flows, heavy UDP flows, and malicious flows. (If the rate of UDP flow is below 100 Kbps, we call it small. If it is between 200 Kbps and 1 Mbps, we call it heavy. If it is more than 1 Mbps, we call it malicious. This is based on the assumption of the bottleneck bandwidth being 1 Mbps.) The network topology is shown in Figure 1 with link bandwidth of 1 Mbps and queue buffer size of 300 packets in the router. Each packet size is 1000 Kb, and the simulation time is 1000 seconds.

Here we created 1000 TCP flows, 30 UDP flows to approach the practical network traffic distribution. These 30 UDP flows are divided to 5 groups as following, they are 10 small users (10 Kbps), 5 little heavy flows (100 Kbps), 5 moderately heavy flows (500 Kbps) based on the statistical UDP rate distribution, and also 10 malicious flows (5 at 1 Mbps, 5 at 10 Mbps). As we know, a VoIP flow occupies about 3–16 Kbps, which is the one of the most popular applications now, and the multimedia streaming flows need around 100–500 Kbps, depending on its frame rate and video resolution. (Compared to the general VoIP 6–8 Kbps, Skype uses P2P techniques to occupy 3–16 Kbps bandwidth to provide better performance.) We monitor the average throughput of these flows and compare the TCP throughput as a percentage of the ideal fair rate.

For Drop-Tail and RED, they both give most of the bandwidth to the heavy UDP users while the TCP flows have nearly zero throughput because they will slow down naturally. Therefore, we do not compare them in the figure. As can be observed in Figure 2. CHOKe 1 cannot protect TCP flows well because of wrong number of drop candidates. Even CHOKe with the right number of dropping candidates, 10, can only provides 50% protection ability, while SFQ has zero-protection ability to the TCP users. (The number of dropping candidates in CHOKe should be the same as that of malicious flows which is indicated in [9]) As we know, SFQ uses queue diversity to provides fairness (divide the FIFO single queue (300 pkts) to 16 buckets (300/16 pkts)), but it cannot prevent the buffer-full problem. The protection ability (fairness) of SFQ is depending on the number of buckets and the queue size of each bucket. AFD can provide 78% protection ability but it needs 30000 buffer size to save the sampled packet history which is a large cost and still UDP flows grab more bandwidth than TCP users. On the contrary, BREATHe, using a simple dropping algorithm with throughput statistics, can effectively provide fair bandwidth allocation. In addition, it bounded every flow's throughput just like scheduling algorithms, but avoided the need for complicated rate estimation and rate shaping methods [1]. In Figure 3, we compare the rate distribution range of these 1000 responsive TCP flows. CHOKe cannot guarantee fairness and neither can AFD. (The more fairness, the more concentrated rate distribution.) But in BREATHe, 70% of the TCP flows are between 0.6 Kbps and 1.2 Kbps, whereas the ideal fair throughput is 0.97 Kbps. Therefore, BREATHe can provide better fairness to TCP flows than other algorithms.

5. Concluding Remarks

In this letter, we proposed a simple packet dropping scheme, BREATHe, which aims to protect responsive TCP flows and approximate fair queueing with acceptable implementation overhead. It has similar advantages as RED in terms of avoiding global synchronization and absorbing bursty traffic. Moreover, it also has the feature to limit all flow's maximum throughput, but avoid full-queue problems when congestion occurs. Besides protecting responsive TCP flows, simulation results show that BREATHe also provides fairness to flows, even with many malicious flows involved. The other advantage of BREATHe is the small requirement on queue size because its main dropping decision is depending on throughput statistics.

References

[1] B. Braden, D. Clark, J. Crowcroft, et al., "Recommendations on queue management and congestion avoidance in the internet," IETF RFC (Informational), 2309, April 1998.

[2] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Proceedings of the ACM Symposium on Communications Architectures & Protocols (ACM SIGCOMM '89)*, pp. 1–12, Austin, Tex, USA, September 1989.

[3] P. E. McKenney, "Stochastic fairness queueing," in *Proceedings of the 9th Annual Joint Conference of IEEE Computer and Communications Societies (INFOCOM '90)*, vol. 2, pp. 733–740, San Francisco, Calif, USA, June 1990.

[4] A. Manin and K. Ramakrishnan, "Gateway congestion control survey," IETF RFC (Informational), 1254, August 1991.

[5] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993.

[6] D. Bertsekas and R. Gallager, *Data Networks*, Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 1992.

[7] D. Lin and R. Morris, "Dynamics of random early detection," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 4, pp. 127–137, 1997.

[8] T. J. Ott, T. V. Lakshman, and L. H. Wong, "SRED: stabilized RED," in *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '99)*, vol. 3, pp. 1346–1355, New York, NY, USA, March 1999.

[9] R. Pan, B. Prabhakar, and K. Psounis, "CHOKe—a stateless active queue management scheme for approximating fair bandwidth allocation," in *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '00)*, vol. 2, pp. 942–951, Tel Aviv, Israel, March 2000.

[10] Y. Jiang, M. Hamdi, and J. Liu, "Self adjustable CHOKe: an active queue management algorithm for congestion control and fair bandwidth allocation," in *Proceedings of the 8th IEEE International Symposium on Computers and Communication (ISCC '03)*, vol. 2, pp. 1018–1025, Antalya, Turkey, June-July 2003.

[11] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker, "Approximate fairness through differential dropping," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 2, pp. 23–39, 2003.

[12] R. M. Karp, S. Shenker, and C. H. Papadimitriou, "A simple algorithm for finding frequent elements in streams and bags," *ACM Transactions on Database Systems*, vol. 28, no. 1, pp. 51–55, 2003.