Stochastic Fair Traffic Management for Efficient and Robust IP Networking

Jae Chung
Airvana, Inc.
Chelmsford, MA 01824, USA
jchung@airvana.com

Mark Claypool, Robert Kinicki
Worcester Polytechnic Institute
Worcester, MA, 01609, USA
{claypool|rek}@cs.wpi.edu

Abstract

As use of non-TCP applications such as streaming media and network games increases, the potential for unfair, misbehaving flows and the threat of congestion collapse also increases. This paper introduces a statistical traffic filtering technique, Stochastic Fairness Guardian (SFG), that effectively regulates misbehaving flows with minimal traffic state information. SFG can be used in conjunction with an active queue management (AQM) mechanism to improve both network protection and efficiency. Simulations are used to evaluated SFG and the integration of SFG with a proportional-integral (PI) controller in comparison with other similar statistical flow management mechanisms including RED-PD, SFB and CHOKe. The SFG-PI combination outperforms other mechanisms in terms of fairness, queuing delay, stability and TCP performance over a wide range of realistic traffic loads and conditions.

1 Introduction

While TCP has end-host congestion control mechanisms designed to manage Internet congestion, TCP can respond slowly to congestion due to the end-host congestion detection that occurs only when router buffers overflow. This wastes network bandwidth and lowers network goodput. The limitations of end-host only congestion control can be relieved by adding active queue management (AQM) [3, 11, 14, 15, 17, 18, 20] with explicit congestion notification (ECN) [28] to network routers. AQM with ECN provides prompt congestion feedback and adds efficiency without requiring packet drops.

Despite the robustness of TCP, emerging Internet applications such as streaming media and network games often use UDP as their transport protocol. As the use of non-TCP applications increases, the Internet must support more flows with improper or no end-to-end congestion control. This trend carries the potential for significant imbalance in the link capacities used by TCP and UDP flows that threatens Internet stability. In the worst case, an extrapolation of this trend could lead to Internet congestion collapse [6].

Router-based approaches to handling unresponsive flows can be generally divided into scheduling-based and preferential-based packet dropping mechanisms. Scheduling-based techniques, such as Fair Queuing (FQ) [10] and Stochastic Fair Queuing (SFQ) [25], allocate a separate queue to each flow or group of flows passing through a router's outgoing link and transmit packets from the queues in round-robin fashion. Scheduling-based mechanisms are generally expensive to implement due to the complexity of the link/packet scheduling. Moreover, it may be undesirable to combine a scheduling-based mechanism with an AQM congestion feedback controller due to the redundancy inherent in providing queue buffers needed to support both mechanisms.

Preferential-based packet dropping techniques monitor, detect and regulate misbehaving flows before forwarding packets to an outbound link queue that may or may not be managed by a separate AQM controller. Preferential-based dropping mechanisms can be further categorized by their complexity and the amount of state information maintained. The most complex mechanisms, including Fair Random Early Drop (FRED) [22], Core Stateless Fair Queuing (CSFQ) [30] and Rainbow Fair Queuing (RFQ) [7], require per-flow state information. Since per-flow state information does not scale well for high capacity networks with many flows, this is a significant weakness for FRED. However, CSFQ and RFQ reduce this problem by requiring per-flow state information only at DiffServ [4]-like edge routers.

Other preferential-based dropping techniques do not require an edge-core architecture for scalability. Random Early Detection with Preferential Dropping (RED-PD) [23], Stochastic Fair Blue (SFB) [12] and CHOKe [26] use statistical flow management to address scalability. RED-PD and SFB employ statistical flow monitoring to identify and regulate misbehaving flows. RED-PD uses RED congestion notification history and SFB employs a Bloom filter [5] to identify potentially misbehaving flows. Although statistical flow monitoring can significantly reduce the flow state information needed to be maintained when a small number of flows account for the majority of the Internet traffic [27], the mechanisms used to identify misbehaving flows are complex and may induce significant processing overhead. While CHOKe does not require any flow state information, the stateless design makes it difficult to achieve target per-flow bitrates and well-behaved flows may be unluckily punished under light traffic loads.

This paper introduces a novel statistical traffic filtering technique, the Stochastic Fairness Guardian (SFG), that regulates misbehaving flows with minimal traffic state information. SFG uses a multi-level hash scheme to place incoming flows into different flow groups at each level and approximates a proper packet drop rate for each flow by monitoring the incoming traffic rates for the groups to which the flow belongs. SFG used in combination with an AQM congestion feedback controller improves both network protection and efficiency. SFG is evaluated in conjunction with the proportional-integral (PI) controller [18]. The combination of SFG and PI (SFG-PI) is compared against RED-PD, SFB and CHOKe, and Drop-Tail queue management through simulations. The results show that SFG-PI outperforms other mechanisms in terms of protection, stability, queuing delay and overall TCP performance under a wide range of traffic mixes. The simulations also demonstrate that SFG with Drop-Tail queuing provides simple and effective fairness protection that complements the weakness of Drop-Tail alone.

The paper is organized as follows: Section 2 describes the design of SFG; Section 3 provides SFG configuration guidelines; Section 4 evaluates the performance of SFG and SFG-PI using simulations; and Section 5 presents conclusions and possible future work.

2 Stochastic Fairness Guardian

Stochastic Fairness Guardian (SFG), a highly scalable statistical traffic filter, uses a small amount of state information to provide stochastically fair network resource allocation and network protection. Using a pre-queue management mechanism, SFG preferentially drops incoming packets in proportion to a flow's approximated unfair resource usage. In SFG, a flow is an abstract entity identified by a combination of source/destination address, protocol and port numbers. While flow monitoring and accounting is challenging for RED-PD [23] and SFB [12] due to determining the lifetime of a flow, SFG does not need to monitor nor account for individual flows to filter traffic. Thus, in the rest of the paper the terms "incoming packet" and "incoming flow" are used interchangeably.

To approximate and regulate unfair network usage, SFG uses a multi-level traffic group management technique. SFG, shown in Algorithm 1, clusters incoming flows into

Algorithm 1 Stochastic Fairness Guardian (SFG)

```
Every d_s seconds:
for i = 0 to L − 1 do
    for j = 0 to N − 1 do
        prob[i][j] ← (bytes[i][j] − d_s C/N)/bytes[i][j];
        bytes[i][j] ← 0;    /* update drop_p for all bins */
    end for
end for
Every packet arrival:
p = 1;
for i = 0 to L − 1 do
    j = hash(i, packet);
    p = min(p, prob[i][j]);  /* take min drop_p seen so far */
    bytes[i][j] ← bytes[i][j] + sizeof(packet);
end for
if (uniform(0, 1) ≤ p) then
    drop(packet);
    return;
end if
queue(packet);
Functions:
  hash(key, packet) Returns hash (<N) for given key and packet.
  drop(packet): Drops the packet.
  queue(packet): Passes the packet to the queue manager.
Variables:
  prob[L][N], bytes[L][N], i, j, p
Parameters:
  C: link capacity (bytes per second)
  L: number of levels
  N: number of bins in a level
  d_s: measurement interval
```

N different traffic groups in each of L levels using an independent hash function for each level. SFG maintains N x L bins, with each bin in a level assigned an equal share $(1/N)$ of the outbound link capacity (C). Every d_s second epoch, SFG computes and updates the packet drop probability for each bin $(prob[i][j])$ by taking the incoming traffic rate of the last measurement epoch $(bytes[i][j]/d_s)$ as an estimate of this epoch's packet arrival rate for the flows in the bin, and setting the drop probability such that no more than C/N capacity is used by a bin.

Upon packet arrival, SFG looks up the packet drop probabilities for the L bins to which the packet belongs and applies the minimum drop probability to the packet. Choosing the minimum drop probability protects TCP flows that share one or more accounting bins with other high bitrate flows. Figure 2 shows an example of SFG selecting drop probabilities for three different flows, where rounded-corner boxes represent the accounting bins and shaded boxes represent the bitrate of each flow. In this example, packets of $flow1$ are dropped with a probability $p = 0.03$, the minimum drop probability of all the bins to which $flow1$ belongs. Similarly, $flow2$ gets $p = 0.02$ and $flow3$ gets $p = 0.00$.

A drawback of using static hash filters for flow group assignments is that a well-behaved flow that unluckily shares all its bins with misbehaving flows can be unfairly treated

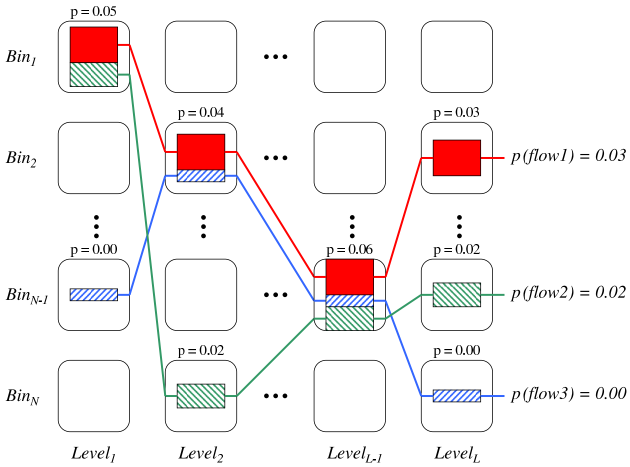

Figure 1. An Example SFG showing three flows. The size of the shaded blocks indicate the flow bitrates. The drop probability applied to each flow is indicated on the right.

for the lifetime of the flow. SFG eases this concern by a simple modification to Algorithm 1 such that two hashes in each level are used, one for the drop probability access for the current epoch and the other for the control data collection for the next epoch. Thus, a flow assigned to polluted bins in all levels in the current epoch can be re-hashed into different bins in the next epoch. This fairness enhancement comes at little additonal cost, since SFG flow group managment for the current epoch is independent of previous epochs.

SFG shares structural similarities with the Bloom filter technique used in SFB [12] in that both mechanisms use multi-level hashing to group flows. However, the major difference is that the Bloom filter in SFB is used as an unresponisve flow identification tool, while SFG uses Bloom-like stochastic fair resouce management to prevent a few misbehaving flows from dominating the outbound link utilization. By periodically updating packet drop probabilities for the bins, SFG inherently has less overhead than SFB with Blue [11] inside each bin where the congestion notification probabilities of the relevent Blue bins are updated for every arriving packet.

Combining SFG with an AQM enhances TCP performance by avoiding packet drops through the AQM while still providing network protection through SFG. The next section provides SFG configuration guidlines for setting L, N and d_s, and addresses issues associated with combining SFG with a queue manager to improve performance.

3 Configuration

This section develops a false positive model to estimate the probability of well-behaved flows being incorrectly identified by SFG as a misbehaving flows. SFG configuration guidelines are provided with a practical SFG integration mechanism that can be applied to both a Drop-Tail queue and an AQM to maximize the benefit of SFG.

An analytic model is developed to determine the false positive flow punishment ratio for SFG, i.e. how often a well-behaving flow is unfairly handled because it shares all of its bins with misbehaving flows. Model parameters include: L - the number of levels supported by SFG , N - the number of bins in each level, and B - the number of misbehaving flows in the system. The first step is to determine the expected number of bins occupied by one or more misbehaving flows (referred to as polluted bins) in a level.

Let $T(B,i)$ be the number of ways to distribute B flows into i bins such that no bin is empty, where $B > i$. This well-understood probability problem is computed as follows:

$$T(B,i) = \sum_{k=0}^{i} (-1)^k \binom{i}{k} (i-k)^B \tag{1}$$

Define $P_w(N,B,i)$ as the probability that exactly i bins from the N total bins are polluted with B misbehaving flows. Computing P_w, requires determining the total number of possible instances of the event. Let $W(N,B,i)$ be the number of ways to pollute exactly i bins from N total bins with B misbehaving flows. This is equal to the number of ways to choose i bins from N total bins and distribute B flows into the chosen i bins such that no bin is empty. Thus, $P_w(N,B,i)$ is determined by dividing $W(N,B,i)$ by the number of ways to put B flows into N bins:

$$P_w(N,B,i) = \frac{W(N,B,i)}{N^B} = \frac{\binom{N}{i} T(B,i)}{N^B}$$

Let $E_w(N,B)$ be the expected number of polluted bins in a level, given N total bins and B misbehaving flows. E_w is then the sum over all possible number of polluted bins times its occurrence probability $P_w(N,B,i)$:

$$E_w(N,B) = \sum_{i=0}^{B} (i \; P_w(N,B,i))$$

Knowing $E_w(N,B)$, the false positive probability, $P_{fp}(L,N,B)$, that a well-behaved flow shares its bins in all levels with misbehaving flows can be computed as:

$$\begin{aligned} P_{fp}(L,N,B) &= \left(\frac{E_w(N,B)}{N}\right)^L \\ &= \left(\frac{1}{N^{B+1}} \sum_{i=0}^{B} i \binom{N}{i} T(B,i)\right)^L \end{aligned} \tag{2}$$

Equation 2 can be used as a secondary SFG configuration tool to find an appropriate number of levels (L) that lowers the false positive error rate after configuring the number of bins per level (N) based on an expected maximum number of misbehaving flows ($\hat{B}$). A misbehaving flow in this context is a flow that is not TCP-friendly [13], where a TCP-friendly flow's data rate does not exceed the maximum rate of a conformant TCP connection under the same network conditions. Once N is determined, the rate limit for misbehaving flow classification becomes apparent, i.e. C/N and $\hat{B}$ can be estimated.

A primary factor in choosing N, the number of bins in a level, is the maximum per-flow bitrate that SFG will permit during congestion. Choosing N directly determines the maximum allowed per-flow bitrate (C/N) for a fixed capacity C. If N is too small, SFG will not filter misbehaving flows that have a low flowrate and will also have a high false positive flow punishment ratio. If N is too large, the small maximum flow rate allowed can affect link utilization at low traffic loads dominated by a few greedy flows and prevent applications with bandwidth requirements larger than C/N from utilizing unused capacity.

One way to address this SFG configuration issue is to only enable SFG when the outbound link is congested, while carefully setting N such that the maximum allowed per-flow rate is small enough to effectively filter misbehaving flows and greater than or equal to a TCP-friendly rate [13] at the SFG enabling/disabling thresholds. This approach offers a static, maximum flow rate during congestion regardless of the actual load level. A more sophisticated approach is to dynamically adjust N every control/measurement epoch using a TCP-friendly rate estimator. The TCP-friendly rate can be estimated using the congestion notification rate (CNR) measured at the router and with the average system round-trip time included as an extra SFG configuration parameter. This dynamic configuration approach is elegant but has increased complexity because the SFG hash functions will have to be adjusted frequently as N changes. This paper explores the feasibility of the simple static on/off approach and leaves the dynamic bin adjustment idea as future work.

To provide an on/off mechanism for SFG, a high/low watermark mechanism (m_h, m_l) for the average CNR estimate is used. To estimate the average CNR, SFG takes a weighted average of the CNR every epoch, where CNR (p_n) is computed as the relational sum of the packet drop rate of SFG (p_d) and the congestion notification probability of the queue manager (p_e):

$$p_n = p_d + (1 - p_d)\, p_e \tag{3}$$

where, p_e is measured in terms of the queue overflow packet loss rate for a Drop-Tail queue, or explicitly reported by the AQM queue manager. For evaluation, SFG is combined with a PI controller [18] (SFG-PI) by taking the CNR computed by PI as p_e.

The SFG configuration process is illustrated by example. Setting $m_h = 0.02$ and $m_l = 0.01$, SFG assumes congestion when CNR is over 2% and under 1%, respectively. The maximum allowed per-flow rate enforced by SFG at congestion can be determined by computing the low boundary TCP-friendly rate at the low watermark using a TCP-friendly rate formula from [13]:

$$T_{tcp} \leq \frac{1.5\sqrt{2/3}\; S}{\tau\sqrt{p_n}} \tag{4}$$

where T_{tcp} is the upperbound TCP-friendly rate, S is average packet size and τ is estimated system round-trip time. By setting $S = 1500$ bytes, a typical MTU, and $\tau = 300$ ms, a value chosen from a valid range of average round-trip times [8, 19], T_{tcp} is 0.5 Mbps. To achieve this maximum allowed per-flow rate, SFG should set $N = 20$ (C/T_{tcp}) for a 10 Mbps output link.

After setting N, the minimum number of levels (L) to provide an optimal false positive error rate is determined using Equation 2 given a range of the expected number of misbehaving flows ($\hat{B}$). $\hat{B}$ can be estimated from existing Internet measurement studies, such as [24] that reports about 10% of the traffic is UDP traffic. Based on this statistic, an average of 1 Mbps UDP traffic is expected for a 10 Mbps link. Assuming all the UDP bandwidth is potentially misbehaving, medium quality video, the typical bitrate will be about 300 Kbps. Thus, $\hat{B}$ is about 3 to 4 misbehaving flows for a 10 Mbps link. If the UDP flows are assumed to be low quality 56 Kbps video streams, a 10 Mbps link may carry as many as 17 misbehaving flows.

Figure 2 plots the false positive error rates of an $N = 20$ system, for $\hat{B} = 1, 5, 10, 15$, to find the number of levels that reduce the per-packet processing overhead from hashing and the false positive error rates. Figure 2 shows that $L = 3$ provides both a low packet processing overhead and a low false positive error rate for the selected range of $\hat{B}$. For example, $P_{fp}(3, 20, 5) \approx 0.01$ and $P_{fp}(3, 20, 10) \approx 0.06$ indicates that the chance that a well-behaving flow is unfairly treated in an epoch by SFG with $L = 3$ and $N = 20$ is about 1% when $\hat{B} = 5$ and about 6% when $\hat{B} = 10$. Similarly, $P_{fp}(3, 20, 15) \approx 0.15$ shows that the chosen SFG setting can also offer relatively low false positive error rates for the higher range of $\hat{B}$.

Assuming each bin requires a 4-byte integer for counting bytes received and a 8-byte double-precision floating number for storing the drop probability, the router memory requirement for a 10 Mbps SFG link with $L = 3$ and $N = 20$ is 720 bytes per output port (3 levels $\times$ 20 bins $\times$ 12 bytes/bin). Similarly, a 10 Gbps link with an equivalent SFG setting of $L = 3$ and $N = 20,000$ requires only 720 Kbytes of router memory per output port.

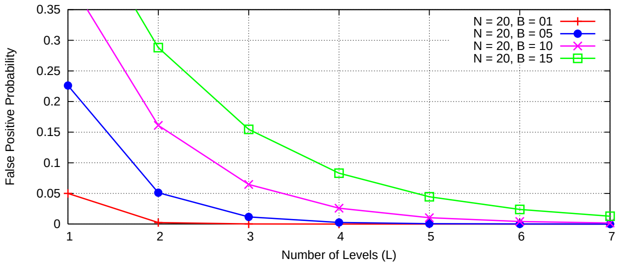

Figure 2. False Positive Probability (N = 20)

The last SFG parameter to discuss is the control/measurement epoch length (d_s). Setting d_s to a couple of seconds (d_s is set to 2 seconds in this investigation), is recommended as this is approximately twice the upper-bound average round-trip time seen on the Internet [8, 19]. This avoids control error due to insufficient control data acquisition and minimizes congestion control interference with the AQM controller. Considering the long flow lifetimes of potentially misbehaving flows such as streaming media and network games, the large epoch length, and hence slow response time is acceptable. A more responsive system would pay a high price in terms of fairness, efficiency and link utilization for packet drops caused by inaccurate SFG control.

4 Evaluation

This section compares the performance of SFG (with Drop-Tail queuing) and SFG-PI (with PI queuing AQM management), with RED-PD [23], SFB [12], CHOKe [26] and Drop-Tail through detailed simulations. The NS simulator is used to model realistic traffic conditions by including long-lived FTP flows (varying in number over time to induce a range of offered loads), background Web traffic, and reverse flows (which can result in ack compression). The NS distribution comes with source code for the PI controller and RED-PD, and makes available the source code for SFB as contributed code. We extended NS to support CHOKe, SFG and SFG-PI.

The simulations utilize a dumbell network topology with a bottleneck link capacity of 10 Mbps and a maximum packet size of 1000 bytes. Based on measurements in [19], round-trip link delays are randomly uniformly distributed over the range [60:1000] ms. The physical queue size at the bottleneck router is fixed at 500 Kbytes, approximately equal to the bandwidth-delay product, in all cases.

The settings for the parameters of the various statistical preferential drop and AQM mechanisms are based on the authors' recommendations. The RED settings are: minimum and maximum thresholds of 50 and 300 respectively, maximum drop probability of 0.15, $W_q = 0.002$, and the gentle option enabled. The additional RED-PD settings include: a target system round-trip time of 100 ms that is used to determine the epoch length for monitoring and for the TCP-friendly rate, flow monitor history window of 5, minimum time to un-monitor a monitored flow and its drop rate threshold of 15 seconds and 0.005, respectively, and maximum drop probability increment step of 0.05. CHOKe, which works in conjunction with RED, is set to divide RED's minimum and maximum queue threshold range into 5 even subregions and apply $2i+1$ drop comparisons for an incoming packet, where $i = \{0, 1, 2, 3, 4\}$ is the subregion ID.

For SFB, the number of levels and bins are set to $L = 3$ and $N = 20$, the unresponsive detection CNP threshold is set to 0.98, and the penalty box time is set to 15 ms. SFB switches hash functions every 20 seconds. For Blue inside each SFB bin, the CNP increment step is 0.005 and the decrement step is 0.001 with a freeze time of 100 ms.

For the PI controller, proportional constant $K_p = 0.71 \times 10^{-5}$ and integral constant $K_i = 2.8116 \times 10^{-5}$ were chosen to meet TCP system stability criteria in [18] with target queue length of $q_0 = 0$. Refer to [9] for the detailed PI congestion controller configuration. For SFG, based on the analysis in the previous section, the on/off thresholds are $m_h = 0.02$ and $m_l = 0.01$, the control/measurement interval $d_s = 2$ seconds, the number of levels $L = 3$ and the number of bins $N = 20$.

All simulations use ECN enabled NewReno TCP for both the long-lived FTP flows and the Web sessions. Each simulation has 50 reverse direction bulk transfer FTP flows and 300 forward direction background Web sessions (using the Webtraf code built into NS) that start evenly distributed during the first 30 seconds. Based on settings from [2, 16], each Web session requests pages with 2 objects drawn from a Pareto distribution with a shape parameter of 1.2 and an average size 5 Kbytes. The Web sessions have an exponentially distributed think time with a mean of 7 seconds, which results in an average utilization of about 2.5 Mbps of the 10 Mbps capacity, a fraction typical of some Internet links [29]. Each simulation has forward direction bulk transfer FTP flows. To test all the mechanisms under dynamic traffic loads, the number of forward direction FTP flows varies every 200 simulation seconds from 10-50-100-200-400 flows and then back down from 400-200-100-50-10 flows.

To more intuitively characterize the degree of congestion experienced by the link beyond simply the number of flows, the Drop-Tail queue simulation with the above network settings was run with only the Web traffic, varying the number of Web session from 1200 to 1800, and recording the packet drop rate for each load. Subsequently, the congested link bandwidth was changed from 10 to 100 Mbps

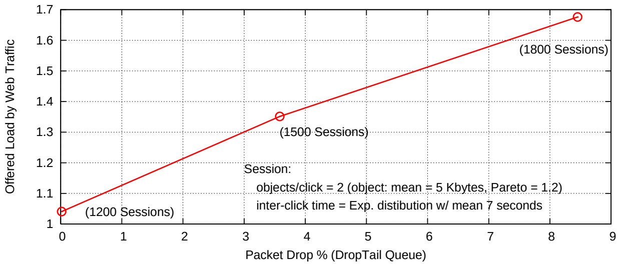

Figure 3. Offered Load by Web Traffic versus Packet Drop Rate (Drop-Tail queue: qlim = 500 Kbytes)

and the simulation re-run to measure the offered traffic rate for each number of Web sessions under a capacity unconstrained condition. The offered traffic rates were then converted to offered loads in relation to the 10 Mbps link capacity, and plotted in relation to the packet drop rates measured for the same number of Web sessions under the 10 Mbps link. Figure 3 shows the linear relationship between the Drop-Tail packet drop rate and offered load.

The offered loads given as a function of the Drop-Tail packet drop rates are useful for characterizing the load created by the TCP traffic mix (i.e. forward direction FTP, backward direction FTP, and background Web traffic), by converting the load of the mixed TCP traffic expressed in terms of the number of FTP flows into the equivalent Web offered traffic load expressed in terms of the packet drop rates of a Drop-Tail queue. Thus, the equivalent offered loads for the TCP traffic mix when the number of forward direction FTP flows is 10, 50, 100, 200 and 400 are about 1.0, 1.1, 1.2, 1.4 and 1.7 respectively. This means, for example, that when the number of FTP flows is 400, the congested link is experiencing about 1.7 times the offered load it can handle without having to drop any packets.

While an offered load of 1.7 is probably beyond any realistic load for most routers on today's Internet, this high load serves as a stress test of the various preferential dropping and AQM mechanisms, showing insight into how they handle the traffic in terms of fairness, throughput, stability, queuing delay, packet and byte loss rate, and Web performance. RED-PD, CHOKe, SFB, SFG and SFG-PI are evaluated using the TCP traffic mix, in comparison with Drop-Tail and PI controlled queue management mechanisms without a preferential dropping mechanism.

4.1 TCP Traffic Mix

This section compares the performance of the various statistical preferential drop mechanisms with that of a Drop-Tail (DT) and PI controller over the range of loads with the TCP traffic mix (ie - no unresponsive flows). Figure 4 shows the queue dynamics (top) and system throughput (bottom) of DT, PI, SFB, CHOKe, RED-PD, SFG and SFG-PI. The byte loss rate, packet drop rate, and average Web object service time for each system is shown in Figure 5

First, comparing the queue dynamics, throughput and byte loss rate of Drop-Tail with the PI controller shows the potential benefits of using AQM: control over link quality of service (QoS) (low queue length) and efficient link utilization. The PI controller is able to stably control traffic over the entire load range, keeping the queue length low at around 100 Kbytes, and maintaining a high link utilization. The PI controller loss rates are low (less than 2%), even at the offered load of 1.7, resulting in a higher goodput than the Drop-Tail system. The low queue length is desirable for interactive Internet applications and the stable queue size can also greatly reduce the buffer size required to achieve a high link utilization [1].

The packet loss rate and the average Web object service time of Drop-Tail and the PI controller show some less well-known performance aspects of ECN when viewed over the range of traffic loads. Although the network efficiency measured in byte loss rate is consistently better for PI, the packet losses for PI, a combination of ECN-incapable SYN packets for the Web traffic and the backward direction TCP ACK packets, are about twice as high as that of Drop-Tail as the traffic load increases beyond an offered load of 1.2. To control traffic at congestion, the PI controller, and more generally any AQM using ECN, must maintain a higher congestion notification probability (CNP) than the packet-drop congestion notification rate of a Drop-Tail queue. As a result, PI, by dropping non-ECN packets with the CNP, favors ECN-capable packets over non-ECN packets especially at high traffic loads, yielding a significantly higher packet drop rate than Drop-Tail for conditions in which small, non-ECN enabled packets dominate.

This phenomenon creates the Web object delivery performance crossover for the Drop-Tail and PI systems as the offered load changes from 1.2 to 1.4, at which point the initial TCP timeout for SYN packet drops becomes the dominating factor for Web object service times. At the peak load of 1.7, the average Web object service time for PI is about 5 seconds, while the Web object service time for Drop-Tail is about 2 seconds. For the traffic load ranges below 1.2, the Web performance results are consistent with the experimental measurement results from [21], showing AQM with ECN can significantly benefit Web traffic at offered loads from 0.9 to 1.0. In contrast, for traffic load ranges above 1.2 or 1.3, Web performance can be significantly degraded by AQM with ECN, although such high loads are uncommon in practice.

The various performance measures of SFG (with Drop-Tail queue management) shown in Figure 4 and Figure 5

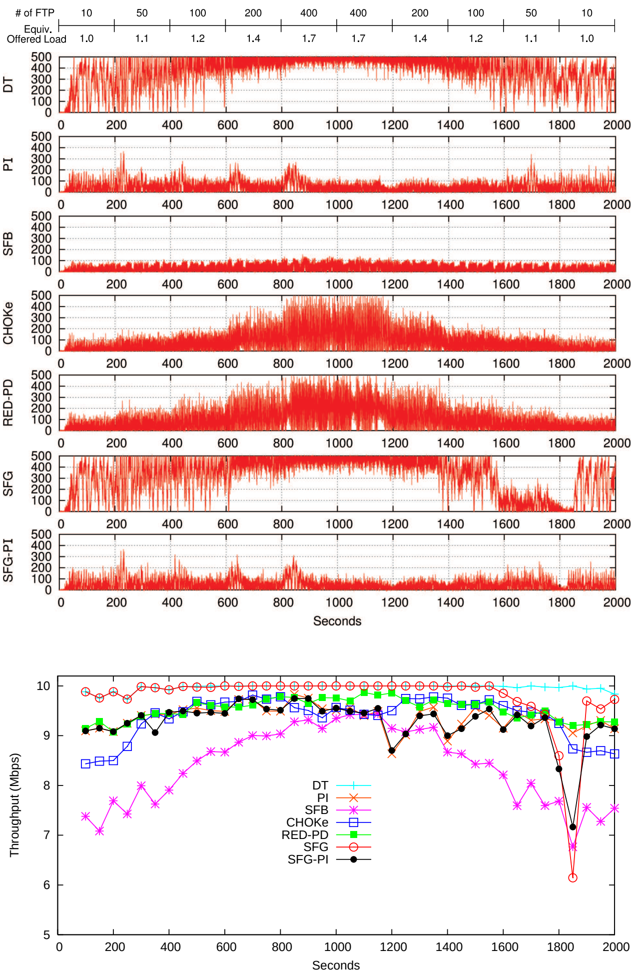

Figure 4. TCP Traffic Mix - Queue Dynamics (top) and Throughput (bottom)

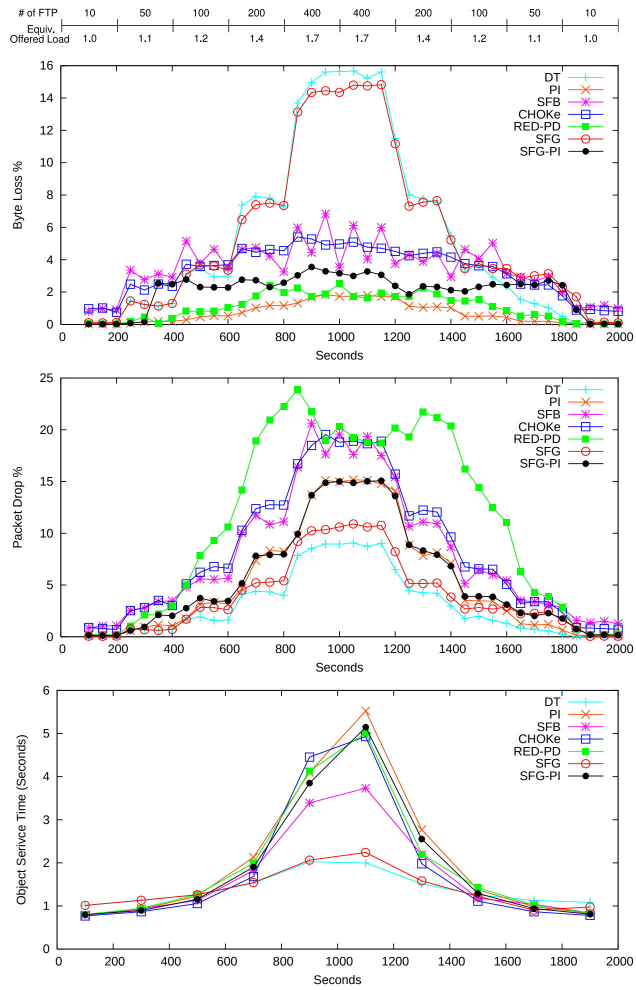

Figure 5. TCP Traffic Mix - Packet Drop Rate (top), Byte Loss Rate (middle) and Average Web Object Service Time (bottom)

closely match those of Drop-Tail, indicating that SFG, activated from 300 to 1900 seconds, works well with Drop-Tail queue management for the TCP traffic mix. Similarly, the performance of SFG-PI closely matches that of the PI controller except for the slightly higher byte loss rates, indicating SFG interferes little with the ability of PI to control TCP traffic. As congestion clears with the termination of the Web flows at about 1850 seconds, SFG and SFG-PI turns off the fairness enforcement mechanism to maximize the link utilization. The adaptation performance can be improved by dynamically adjusting the configuration of N as a replacement for the on/off mechanism, as discussed in Section 3.

Comparing the performance of SFB, CHOKe and RED-PD with that of SFG-PI in Figure 4 and Figure 5, SFB, CHOKe and RED-PD have consistently higher packet drop rates and byte loss rates than SFG-PI, except for RED-PD's slightly lower byte loss rate caused by RED-PD's higher operating queue length. Yet CHOKe, which works in conjunction with a RED controller, is not able to benefit from this higher RED queue length due to its rather inefficient statistical preferential dropping mechanism. CHOKe has a low average throughput of 8.5 Mbps under the low offered loads during the first and last 200 seconds. Throughout the simulation, SFB suffers from low link utilization caused by the inefficient Blue rate control for the traffic mix.

4.2 Unresponsive, High-Bitrate Flows

To evaluate the performance of the various preferential dropping mechanisms, this section considers simulations where five 2 Mbps CBR UDP flows are added to the TCP

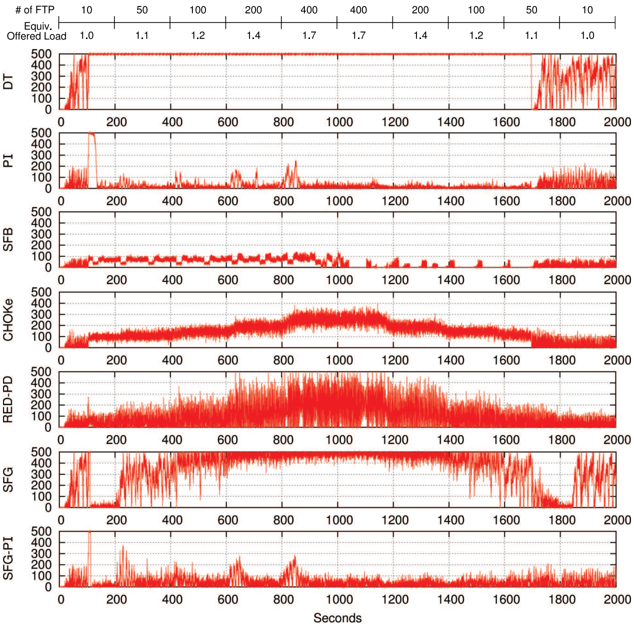

Figure 6. Unresponsive High-Bitrate CBR Flows - Queue Dynamics

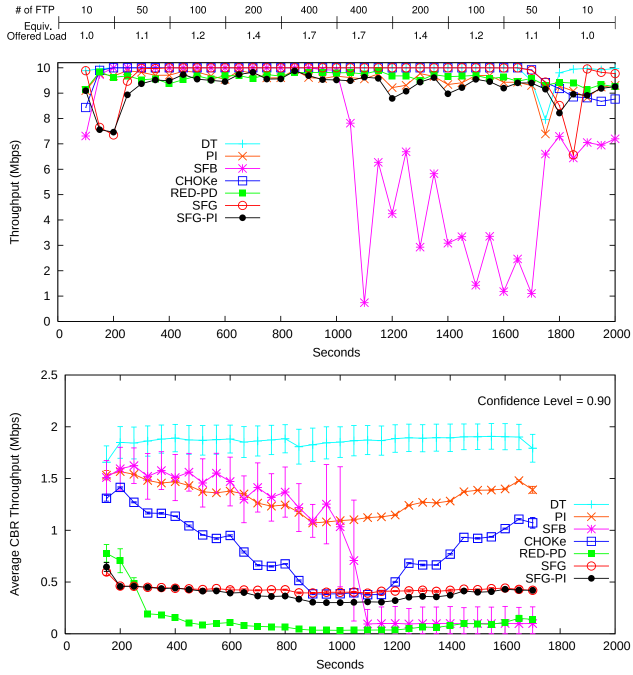

Figure 7. Unresponsive High-Bitrate CBR Flows - System Throughput (top) and CBR Throughput (bottom)

traffic mix used in Section 4.1, at time 100 seconds and stopped at time 1700 seconds. For comparison, the performance of Drop-Tail and the PI controller are examined to determine the impact of the high-bitrate unresponsive CBR flows. Figure 6 shows the queue dynamics, Figure 7 shows the system throughput (top) and the average throughput of the five unresponsive CBR streams (bottom), and Figure 8 shows the average Web object service times.

Figure 6 shows that the Drop-Tail queue remains full for the duration of the high-bitrate CBR flows and Figure 7 (bottom) demonstrates the unfairness of Drop-Tail whereby the CBR flows utilize about 95% of the link capacity with the average throughput of the CBR flows close to 2 Mbps. The average Web service time for Drop-Tail ranges from about 50 to 300 seconds, too high to be seen in Figure 8.

PI controls the aggregated traffic and the unresponsive flows better than Drop-Tail by applying a high CNP that drops UDP packets while marking the ECN-enabled TCP packets. As shown in Figure 6, the PI controller keeps the queue length consistently low while maintaining a high link utilization even in the supersaturated conditions. However, like Drop-Tail, PI is unfair, and the Web traffic experiences high service times, ranging from about 2 to 14 seconds throughout the simulation.

Figure 7 (bottom) shows SFB failing to detect the unresponsive UDP flows until the offered load reaches 1.7. SFB is even more unfair than PI. Moreover, when SFB finally detects the five unresponsive streams and restricts their rate by putting them into a penalty box, there is significant link underutilization, since SFB fails to lower the congestion notification probability (CNP) accordingly. SFB's failure to properly adjust the CNP when there is an increase in the available capacity is also apparent in the Web object service time, shown in Figure 8, that is similar to or larger than that of PI for the second half of the simulation.

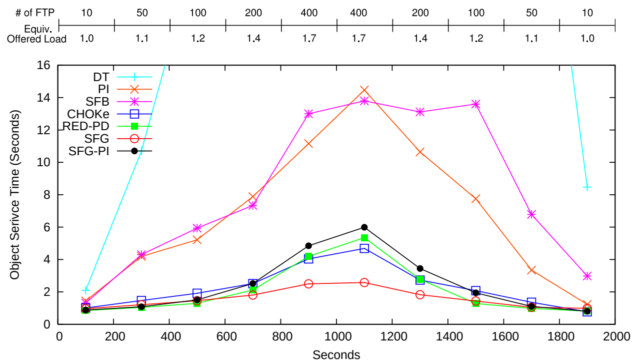

Figure 8. Unresponsive High-Bitrate CBR Flows - Average Web Object Service Time

Using its statistical filtering mechanism, CHOKe regulates the unresponsive, high-bitrate flows. However, CHOKe's fairness is coarse because CHOKe heuristically increases the number of random match drops for each incoming packet as the load increases. RED-PD is able to effectively regulate the high-bitrate CBR flows by monitoring and then restricting the flows to no more than the periodically adjusted TCP-friendly rate. As observed from the queue dynamics of both CHOKe and RED-PD, by frequently adjusting the maximum allowed flow rate for the CBR flows, RED-PD affects the stability of the RED and causes the RED-PD queue to oscillate more than the CHOKe queue.

Both SFG with Drop-Tail and SFG-PI effectively restrict the rate of the unresponsive high-bitrate CBR flows to the target maximum of 0.5 Mbps. For some additional complexity, the fairness control of SFG could be enhanced to the level of RED-PD by dynamically adjusting the target flow rate, as briefly discussed in Section 3 (and left as future work). However, even without this adjustment, the main goal of SFG and SFG-PI has been met, and that is not to strictly enforce TCP-Friendly fairness but rather to provide reasonable protection from egregiously unresponsive flows.

Finally, the consistently stable and low queue dynamics in Figure 4 (top) of and Figure 6 show that the statistical filtering mechanism of SFG does not noticeably affect the congestion control of the PI controller. Providing protection for TCP flows against the unresponsive flows while maintaining the stability of the queue manager is the key contribution of SFG.

The performance of SFG and SFG-PI was also tested with more realistic MPEG-like unresponsive variable bit rate flows. Similarly to the results presented in this section, SFG-PI outperforms other mechanisms in terms of protection of TCP flows and stable and low queue length. Due to space constraint, the results are omitted from this paper, but presented in [9].

5 Conclusions and Future Work

This paper presents Stochastic Fairness Guardian (SFG), a statistical filter that precedes a router queue manager and preferentially drops packets to protect responsive flows from unresponsive flows. SFG can be used with Drop-Tail queuing or with an AQM to improve efficiency and provide protection. This paper also develops an analytic model for estimating the chance of a flow being incorrectly limited with SFG and provides practical SFG configuration guidelines through performance bottleneck analysis and false positive rate analysis.

SFG integrated with Drop-Tail and SFG with a PI controller (SFG-PI) are evaluated via simulation and compared against other preferential drop mechanisms including SFB [12], CHOKe [26] and RED-PD [23], and also compared to PI and Drop-Tail queuing with no filtering mechanisms. Performance metrics include queue dynamics, throughput, fairness, byte loss rate, packet drop rate and Web object service time. Considering overall performance and design complexity, SFG-PI outperforms other preferential dropping mechanisms and Drop-Tail and PI over a wide, practical range of traffic loads. SFG provides protection for the responsive flows against the unresponsive flows in all the tested traffic load and conditions without compromising the stability of the queue manager.

Future work includes extending SFG to dynamically adjust the number of bins per level (N) each epoch such that the maximum allowed flow rate imposed by SFG is set to an estimate of the TCP-friendly rate of the system. Additional future work is to implement SFG and SFG-PI into the Linux kernel and measure the filtering overhead.

References

[1] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing Router Buffers. In *Proceedings of ACM SIGCOMM*, Portland, OR, USA, Sept. 2004.

[2] M. Arlitt and T. Jin. Workload Characterization of the 1998 World Cup Web Site. Technical Report HPL-1999-35R1, HP Laboratories Palo Alto, Sept. 1999.

[3] S. Athuraliya, V. H. Li, S. H. Low, and Q. Yin. REM: Active Queue Management. *IEEE Network*, May 2001.

[4] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Services. *IETF Request for Comments (RFC) 2475*, Dec. 1998.

[5] B. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7), July 1970.

[6] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on Queue Management and Congestion Avoidance in the Internet. RFC-2309, Apr. 1998.

[7] Z. Cao, Z. Wang, and E. Zegura. Rainbow Fair Queuing: Fair Bandwidth Sharing Without Per-Flow State. In *Proceedings of IEEE Infocom*, pages 922 – 931, Tel-Aviv, Israel, Mar. 2000.

[8] B.-Y. Choi, S. Moon, Z.-L. Zhang, K. Papagiannaki, and C. Diot. Analysis of Point-To-Point Packet Delay in an Operational Network. In *Proceedings of IEEE INFOCOM*, Hong Kong, Mar. 2004.

[9] J. Chung. *Congestion Control for Streaming Media*. PhD thesis, Worcester Polytechnic Institute, Oct. 2005. Committee: Professor Mark Claypool, Professor Robert Kinicki, Professor Craig Wills, and Professor Kevin Jeffay (UNC).

[10] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *Proceedings of ACM SIGCOMM*, Austin, TX, USA, Sept. 1989.

[11] W. Feng, D. Kandlur, D. Saha, and K. Shin. Blue: An Alternative Approach To Active Queue Management. In *Proceedings of the Workshop on Network and Operating Sys-*

tems Support for Digital Audio and Video (NOSSDAV), June 2001.

[12] W. Feng, D. Kandlur, D. Saha, and K. Shin. Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness. In *Proceedings of IEEE Infocom*, pages 1520 – 1529, Anchorage, Alaska, USA, Apr. 2001.

[13] S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. *IEEE/ACM Transactions on Networking*, Feb. 1999.

[14] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, Aug. 1993.

[15] Y. Gao and J. Hou. A State Feedback Control Approach to Stabilizing Queues for ECN-Enabled TCP Connections. In *Proceedings of IEEE INFOCOM*, San Francisco, CA, USA, Apr. 2003.

[16] F. Hernandez-Campos, K. Jeffay, and F. Smith. Tracing the Evolution of the Web Traffic: 1995-2003. In *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Oct. 2003.

[17] Z. Heying, L. Baohong, and D. Wenhua. Design of a Robust Active Queue Management Algorithm Based on Feedback Compensation. In *Proceedings of ACM SIGCOMM*, Karlsruhe, Germany, Aug. 2003.

[18] C. Hollot, V. Misra, D. Towsley, and W. Gong. On Designing Improved Controllers for AQM Routers Supporting TCP Flows. In *Proceedings of IEEE Infocom*, Anchorage, AK, USA, Apr. 2001.

[19] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring TCP Connection Characteristics Through Passive Measurements. In *Proceedings of IEEE INFOCOM*, Hong Kong, Mar. 2004.

[20] S. Kunniyur and R. Srikant. Analysis and Design of an Adaptive Virtual Queue. In *Proceedings of ACM SIGCOMM*, San Diego, CA, USA, Aug. 2001.

[21] L. Le, J. Aikat, K. Jeffay, and F. D. Smith. The Effects of Active Queue Management on Web Performance. In *Proceedings of ACM SIGCOMM*, Karlsruhe, Germany, Aug. 2003.

[22] D. Lin and R. Morris. Dynamics of Random Early Detection. In *Proceedings of ACM SIGCOMM Conference*, Cannes, France, Sept. 1997.

[23] R. Mahajan, S. Floyd, and D. Wetherall. Controlling High-Bandwidth Flows at the Congested Router. In *Proceedings of the 9th International Conference on Network Protocols (ICNP)*, Nov. 2001.

[24] S. McCreary and K. Claffy. Trends in Wide Area IP Traffic Patterns: A View from Ames Internet Exchange. In *Proceedings of ITC Specialist Seminar on IP Traffic Measurement, Modeling and Management*, Monterey, CA, USA, Sept. 2000.

[25] P. McKenny. Stochastic Fairness Queueing. In *Proceedings of IEEE Infocom*, San Francisco, CA, USA, June 1990.

[26] D. Mitra, K. Stanley, R. Pan, B. Prabhakar, and K. Psounis. CHOKE, A Stateless Active Queue Management Scheme for Approximating Fair Bandwidth Allocation. In *Proceedings of IEEE Infocom*, Tel-Aviv, Israel, Mar. 2000.

[27] K. Papagiannaki, N. Taft, and C. Diot. Impact of Flow Dynamics on Traffic Engineering Design Principles. In *Proceedings of IEEE INFOCOM*, Hong Kong, China, Mar. 2004.

[28] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC-3168, Sept. 2001.

[29] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy. An Analysis of Internet Content Delivery Systems. In *Usenix Operating Systems Design and Implementation (OSDI)*, pages 315 – 327, Boston, MA, USA, Oct. 2002.

[30] I. Stoica, S. Shenker, and H. Zhang. *Core*-Stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocations in High Speed Networks. In *Proceedings of ACM SIGCOMM Conference*, pages 118 – 130, Vancouver, British Columbia, Canada, Sept. 1998.