Stochastic Fair Aggregate Rate Control: Practical Traffic Management for Efficient and Robust IP Networking

Jae Chung, Mark Claypool and Robert Kinicki
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA 01609, USA
{goos|claypool|rek}@cs.wpi.edu

Abstract

The robustness of the Internet today depends upon the end-to-end congestion control mechanisms of TCP. As use of non-TCP applications such as streaming media and network games grows, the potential for unfair network resource allocation and the threat of congestion collapse increase. The Internet needs a practical solution to protect well-behaving flows from misbehaving flows and avoid collapse. This paper introduces a novel 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. Through simulation, this paper evaluates SFG and the integration of SFG with a previously developed Aggregate Rate Controller (ARC) for TCP traffic in comparison with other similar statistical flow management mechanisms including RED-PD, SFB and CHOKe. Our results show that overall, SFG with ARC 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

TCP, the de-facto Internet transport protocol, has end-host congestion control mechanisms designed to manage Internet congestion. However, TCP can be slow in responding to congestion due to the fact that end-hosts normally detect congestion only when router buffers overflow, wasting network bandwidth and lowering network goodput. The limitations of end-host only congestion control can be relieved by adding active queue management (AQM) [4, 12, 14, 16, 18, 19, 21] with explicit congestion notification (ECN) [29] to network routers. AQM with ECN can provide congestion feedback information promptly and efficiently to the end-hosts without requiring a packet drop.

Despite the robustness of TCP, emerging Internet applications such as streaming media and network games tend to use UDP as their transport protocol. As the use of non-TCP applications increases, the Internet must deal with more flows with improper or no end-to-end congestion control. This trend carries the potential for a significant imbalance in the link capacities used by TCP and UDP flows. This imbalance threatens Internet stability and, in the worst case, an extrapolation of this trend could lead to Internet congestion collapse [7].

Approaches to protect the Internet from potential misuse by unresponsive flows can be divided into scheduling-based and preferential-based packet dropping mechanisms. Scheduling-based techniques, such as Fair Queuing (FQ) [11] and Stochastic Fair Queuing (SFQ) [26], 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) [23], Core Stateless Fair Queuing (CSFQ) [32] and Rainbow Fair Queuing(RFQ) [8], require per-flow state information. The fact that per flow state information does not scale for high capacity networks with many flows is a significant weakness for FRED. However, CSFQ and RFQ reduce this problem by requiring per-flow state information only at DiffServ [5]-like edge routers.

Other preferential-based dropping techniques do not require an edge-core architecture for scalability. Techniques such as Random Early Detection with Preferential Dropping (RED-PD) [24], Stochastic Fair Blue

(SFB) [13] and CHOKe [27], use statistical flow management to address scalability. RED-PD and SFB employ statistical flow monitoring to identify and then regulate misbehaving flows. RED-PD uses the congestion notification history of RED, and SFB uses a Bloom filter [6] to identify potentially misbehaving flows to monitor. Although statistical flow monitoring mechanisms 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 [28], the mechanisms used to identify misbehaving flows are complex and may induce significant processing overhead. To avoid the complexity of flow identification, CHOKe uses a stateless statistical traffic filtering technique that does not require any flow state information. For each incoming packet, CHOKe randomly selects a packet from the output queue and drops both packets both if they are from the same flow. Yet, CHOKe's stateless design makes it difficult to configure the target per-flow bitrate under changing traffic loads. Furthermore, CHOKe may punish well-behaved flows that are unluckily selected and noticeably degrade TCP performance under light traffic loads.

This paper introduces a novel statistical traffic filtering technique, called the Stochastic Fairness Guardian (SFG), that can effectively regulate misbehaving flows with minimal traffic state information. SFG uses a multi-level hash scheme that places 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 can be used in conjunction with a Drop-Tail queue as an effective network protection mechanism. When SFG is used in combination with an AQM congestion feedback controller, the combination can improve both network protection and efficiency. When TCP traffic is effectively controlled by the AQM, the interference between SFG and the AQM for TCP traffic can be minimized such that SFG serves only as a traffic filter for misbehaving, unresponsive flows.

In this work, SFG is evaluated in conjunction with a previously developed AQM mechanism, the Aggregate Rate Controller (ARC) [1]. ARC is a reduced-parameter proportional-integral (PI) controller for TCP traffic designed to support a wide range of traffic conditions. The combination of SFG and ARC, referred to as Stochastic Fair ARC (SFA), is compared against other statistical flow management approaches including RED-PD, SFB and CHOKe, and Drop-Tail queue management through simulations. The results show that SFA outperforms other mechanisms in terms of protection, stability, queuing delay and overall TCP performance under a wide range of realistic traffic mixes and loads that includes a few high bitrate CBR flows and many MPEG-video like VBR streams. The simulations also demonstrate that SFG with Drop-Tail queue management provides simple and effective fairness protection that complements the weakness of Drop-Tail alone.

Algorithm 1 Aggregate Rate Controller (ARC)

Every d_a seconds:

$p \leftarrow p + \alpha(b - \gamma(d_a C - (q - q_0)))$;
$b \leftarrow 0$;

Every $packet$ arrival:

if $(uniform(0,1) \leq p)$ **then**
  if $(mark(packet) == false)$ **then**
    $drop(packet)$;
    $return$;
  end if
end if
$b \leftarrow b + sizeof(packet)$;
if $(enqueue(packet) == false)$ **then**
  $drop(packet)$;
end if

Functions:

$mark(packet)$: ECN mark the packet. Return $false$ on error.
$enqueue(packet)$: Enqueue the packet. Return $false$ if queue full
$drop(packet)$: Drop the packet.

Variables:

p, q, b

Parameters:

C: link capacity (bytes per second)
γ: target link utilization ($\gamma = C_0/C$)
q_0: target queue length in bytes
d_a: measurement interval
α: TCP congestion feedback constant

The remainder of this paper is organized as follows: Section 2 provides background on ARC; Section 3 describes the design of SFG and SFA; Section 4 develops an error model to predict the number of false positives for SFG and provides SFG configuration guidelines; Section 5 uses simulations to evaluate the performance of SFG and SFA along with other statistical flow management approaches; Section 6 describes related work; and Section 7 presents conclusions and considers future work.

2 Background

The proportional-integral (PI) controller is the most widely used controller in modern control systems due to its simplicity and effectiveness. Recently, several AQM researchers [4, 19, 21] have viewed TCP congestion control as a time-delayed feedback control system and thereby were able to develop a PI controller for routers handling TCP flows. While these approaches are promising because a PI controller can provide stability over a wide range of TCP conditions, a critical deployment challenge is the configuration of the PI control parameters in a time-delayed feedback system (i.e., the Internet). There exist no simple and effective PI control parameter configurations for time-delayed systems [31].

As a solution to the difficulty in configuring PI controllers, [1] proposes to carefully reduce the PI parameters, based on classic control theory and a sound understanding of PI behavior for the Internet traffic control domain. Aggregate Rate Controller (ARC), shown in Algorithm 1, is a rate-based implementation of a reduced-parameter PI congestion feedback controller for TCP traffic designed to ease the controller configuration for a wide range of traffic conditions and to offer flexible quality of service (QoS) tuning for the outgoing link.

PI control-based AQM mechanisms determine congestion notification probability (p) for aggregated traffic proportional (α) to the length of a virtual queue. The virtual queue length is computed after reserving some of the link capacity to drain/fill a portion (β) of outbound queue to maintain a target queue length (q_0). ARC reduces the PI control parameters by fixing the queue control parameter β to 1, implying ARC conservatively reserves enough capacity to drain all the packets in the queue due to control error from previous measurement epochs (d_a). This independent PI control variable reduction eases the ARC tuning process while preserving the effectiveness of the PI controller.

ARC provides a practical configuration guideline by modeling a TCP-ARC feedback control system and performing a classical stability analysis using the model. It is shown from the analysis that given a system boundary condition described by the minimum expected number of TCP flows ($\check{N}$) and the maximum expected round-trip time of the system ($\hat{\tau}$), the TCP-ARC system is stable if $\frac{\alpha}{d_a}$ meets the following condition:

$$\left\{ \frac{\alpha}{d_a} \mid \mu(\omega_g, \frac{\alpha}{d_a}) < -6 \right\} \quad and$$
$$\left\{ \frac{\alpha}{d_a} \mid -150^\circ < \phi(\omega_p(\frac{\alpha}{d_a})) < -120^\circ \right\} \quad (1)$$

where, α is the reduced PI control parameter, d_a is the control data measurement interval and:

$$\mu(\omega, \frac{\alpha}{d_a}) = 20 \log_{10} \left(\frac{\frac{\alpha \hat{\tau}^3 \gamma^3 C^3 (1+\gamma)}{4 d_a \check{N}^2} \sqrt{\left(\frac{\hat{\tau}\omega}{1+\gamma}\right)^2 + 1}}{\omega \sqrt{\left(\frac{\hat{\tau}^2 \gamma C \omega}{2\check{N}}\right)^2 + 1} \sqrt{(\hat{\tau}\omega)^2 + 1}} \right)$$

$$\phi(\omega) = \tan^{-1}\left(\frac{\hat{\tau}\omega}{1+\gamma}\right) - \tan^{-1}\left(\frac{\hat{\tau}^2 \gamma C \omega}{2\check{N}}\right) - \tan^{-1}(\hat{\tau}\omega) - \frac{180^\circ}{\pi}\hat{\tau}\omega - 90^\circ$$

ARC also recommends setting the measurement interval (d_a) greater than or equal to the maximum expected round-trip time the system expects to support but smaller than 2 seconds, and then determining α. By choosing a large value of d_a, ARC can reduce control information processing overhead, and more importantly, make the TCP system stable even under unusually large transmission delays or extremely low traffic load.

Algorithm 2 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

3 Stochastic Fairness Guardian

Stochastic Fairness Guardian (SFG) is a highly scalable statistical traffic filter that 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. SFG can be employed either with a Drop-Tail queue or with an AQM mechanism.

SFG defines a flow as an abstract entity that can be identified by a combination of source/destination address, protocol and port numbers. A closely related issue that makes flow monitoring and accounting challenging for approaches such as RED-PD [24] and SFB [13] is determining the lifetime of a flow. However, 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 2, clusters incoming flows into N different traffic groups in each of L levels using an independent hash function for each level. Thus, SFG maintains N x L bins, where each bin in a level is assigned an equal share ($1/N$) of the outbound

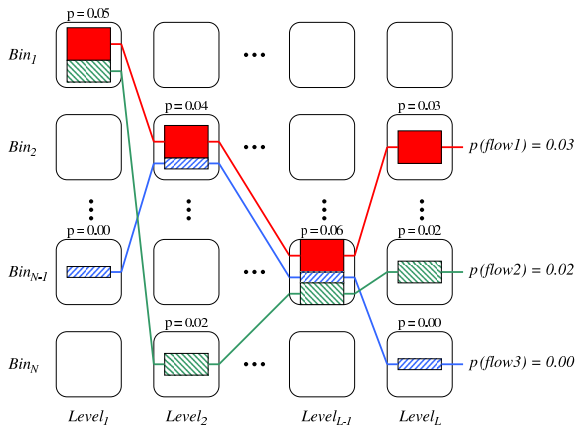

Figure 1: An Example SFG showing three flows. The size of the shaded blocks indicate the flow bitrates. The drop probability applied to each flow is indicated on the right.

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 measurment 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.

When a packet arrives, 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. The motivation behind choosing the minimum drop probability is to protect TCP flows that share one or more accounting bins with other high bitrate flows. Figure 3 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 $flow1$ is dropped with a probability $p = 0.03$ since it is the minimum drop probability of all the bins to which the flow belongs. Similarly, $flow2$ gets $p = 0.02$ and $flow3$ gets $p = 0.00$.

A potential drawback of using static hash filters for flow group assignments is that a well-behaving flow that unfortunately shares all its bins with misbehaving flows can be unfairly treated for the lifetime of the flow. SFG eases this concern by a simple modification to Algorithm 2 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. In this way, a flow assigned to poluted bins in all levels in the current epoch can be re-hashed into different bins in the next epoch. This additional fairness enhancement is easily added to SFG, since SFG flow group managment for the current epoch is independent of previous epochs.

SFG can be used both with a drop-trail queue or with an AQM. The combinaton of SFG with an AQM can enhance TCP performance by avoiding packet drops through the AQM while still providing network protection through SFG. To maximize the benefit of an SFG and AQM combintaion, a careful configuration of SFG and the AQM is required. Note that although SFG is not designed to be a congestion feedback controller for TCP traffic, it may perform the roll of an implicit (packet dropping) congestion feedback controller when faced with fewer than N bandwdith-hungry TCP flows, or when SFG is configured to offer lower link utilization than the following queue manager can offer. Under such circumstances, SFG may result in underutilization, or may degrade TCP performance or even degrade TCP congesion system stability by interfering with the AQM congestion feedback control. 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 maximize preformance benefits.

SFG shares structural similarities with the Bloom filter technique used in SFB [13] 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 accounting bins, SFG inherently has less overhead than does SFB with the Blue AQM [12] inside each accounting bin where the congestion notification probabilities of the relevent Blue bins are updated for every arriving packet.

4 Configuration

This section develops a false positive model to estimate the probability of well-behaving flows being incorrectly identified by SFG as one of the misbehaving flows. Based on the model and performance considerations, 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 potential 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. Parameters in the model 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 is a well-understood probability problem that can be computed as follow:

$$T(B,i) = \sum_{k=0}^{i} (-1)^k \binom{i}{k} (i-k)^B \qquad (2)$$

Let $P_w(N,B,i)$ be 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. Thus, we have:

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

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 can be computed as the sum of each possible outcome 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-behaving flow shares its bins in all levels with misbehaving flows and thus can be unfairly treated can be computed as:

$$\begin{aligned} P_{fp}(L,N,B) &= \left(\frac{E_w(N,B)}{N}\right)^L \\ &= \left(\frac{1}{N^{B+1}} \sum_{i=0}^{B} i \binom{N}{i} T(B,i)\right)^L \end{aligned} \qquad (3)$$

Equation 3 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, defined in this context, is flow that is not TCP-friendly [15] flow, where a TCP-friendly flow is a flow with a data rate that does not exceed the maximum rate of a conformant TCP connection under the same network conditions. In practice it is difficult to determine if a flow is TCP-friendly. For example, a relatively low bitrate unresponsive flow that is classified as a TCP-friendly flow under light traffic loads can turn into a TCP-unfriendly flow at a higher load. Yet, the above definition is sufficient for the purposes of SFG. 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 performance factor to consider 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 effectively filter misbehaving flows that have a low flowrate and will also have a high false positive flow punishment ratio. On the other hand, if N is too large, the small maximum allowed per-flow rate 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 [15] at the SFG enabling/disabling thresholds. This simple approach offers a static, maximum allowed per-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 a TCP-friendly rate formula where the congestion notification rate (CNR) is measured at the router and the average system round-trip time is 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. The CNR estimate at the router is considered as a measure of the congestion level. To estimate the average CNR, SFG takes a weighted average on 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):

$$p_n = p_d + (1 - p_d)\, p_e \qquad (4)$$

where, p_e can be measured in terms of the queue overflow packet loss rate for a Drop-Tail queue, or explicitly reported by the AQM queue manager.

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 [15]:

$$T_{tcp} \leq \frac{1.5\sqrt{2/3}\,S}{\tau\sqrt{p_n}} \quad (5)$$

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 [9, 20], 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 configuring N, the minimum number of levels (L) that can provide an optimal false positive error rate can be determined using Equation 3 given a range of the expected number of misbehaving flows ($\hat{B}$). A reasonable $\hat{B}$ can be estimated based on an Internet measurement study [25] that reports about 10% of the traffic is UDP traffic. Based on this statistics, an average of 1 Mbps UDP traffic is expected for a 10 Mbps link. Assuming all 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. Assuming the UDP flows are low quality 56 Kbps video streams, a 10 Mbps link may carry as many as 17 misbehaving flows.

Figure 4 plots the false positive error rates of an $N = 20$ system, varying L for $\hat{B} = 1, 5, 10, 15$, to find the number of levels that reduces the per-packet processing overhead from hashing as well as the false positive error rates. Figure 4 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 chances 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 $\hat{B}$.

Consider now the router memory requirement for SFG. 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 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 memory per output port.

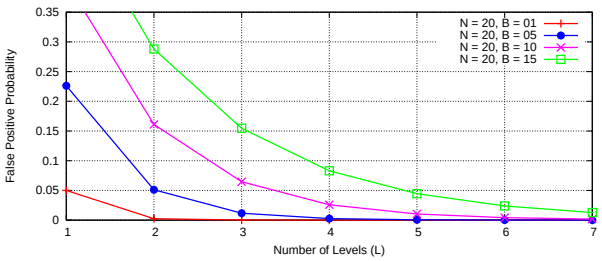

Figure 2: False Positive Probability (N = 20)

The last SFG parameter to discuss is the control/measurement epoch length (d_s). We recommend setting d_s to a couple of seconds (d_s is set to 2 seconds in this investigation), such that it is approximately twice the upperbound average round-trip time seen on the Internet [9, 20]. 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.

5 Evaluation

This section compares the performance of SFG (with Drop-Tail queue management) and the combination of SFG and ARC, referred to as Stochastic Fair ARC (SFA), with RED-PD [24], SFB [13], CHOKe [27] and Drop-Tail through detailed simulations. The simulations attempt to incorporate realistic traffic conditions by including long-lived FTP flows (that vary in number over time to induce a range of offered loads), background Web traffic, and 2-way traffic (which can result in ack compression). The IP packet simulator NS was used for all simulations. The NS distribution comes with source code for RED-PD and makes available the source code for SFB as contributed code. We extended NS to support CHOKe, ARC, SFG and SFA.

The simulations model a dumbell network topology with a bottleneck link capacity of 10 Mbps and a maximum packet size of 1000 bytes. Round-trip link delays are randomly uniformly distributed over the range [60:1000], based on measurements in [20]. The physical queue limit for each AQM and the Drop-Tail queue is set to 500 Kbytes, approximately equal to the bandwidth-delay product for the mean round-trip time.

The settings for the parameters of the various statistical preferential drop and AQM mechanisms are based on the recommendations of their authors (see Related Work in Section 6 for details on other preferential-based

dropping mechanisms). The settings for RED have minimum and maximum thresholds of 50 and 300 respectively, maximum drop probability of 0.15, weighted average factor $W_q = 0.002$, 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 is set to switch hash functions every 20 seconds. For the Blue AQM 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.

The settings for ARC include the measurement epoch $d_a = 1$ second, $\alpha = 1.42 \times 10^{-5}$, the target utilization $\gamma = 0.98$ and the target queue $q_0 = 0$. For SFG, based on the analysis made 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-live FTP flows and the Web sessions. Each simulation has 50 backward 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 [3, 17], 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, such as in [30]. Each simulation has forward direction bulk transfer FTP flows. To test the various mechanisms under different 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

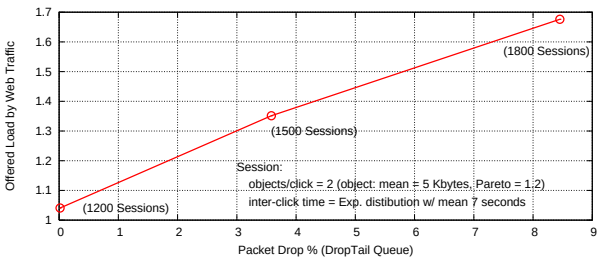

Figure 3: Offered Load by Web Traffic versus Packet Drop Rate (Drop-Tail queue: qlim = 500 Kbytes)

recording the packet drop rate for each load. Subsequently, the congested link bandwidth was changed from 10 to 100 Mbps 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 5 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 SFA are evaluated using the TCP traffic mix, in comparison with Drop-Tail and ARC, queue management mechanisms without a preferential dropping mechanism. The next set of simulations include one unresponsive 10 Mbps CBR flow added to the TCP traffic mix, replacing the one unresponsive flow with five unresponsive 2 Mbps CBR flows, and finally replacing the five unresponsive flows with five unresponsive MPEG

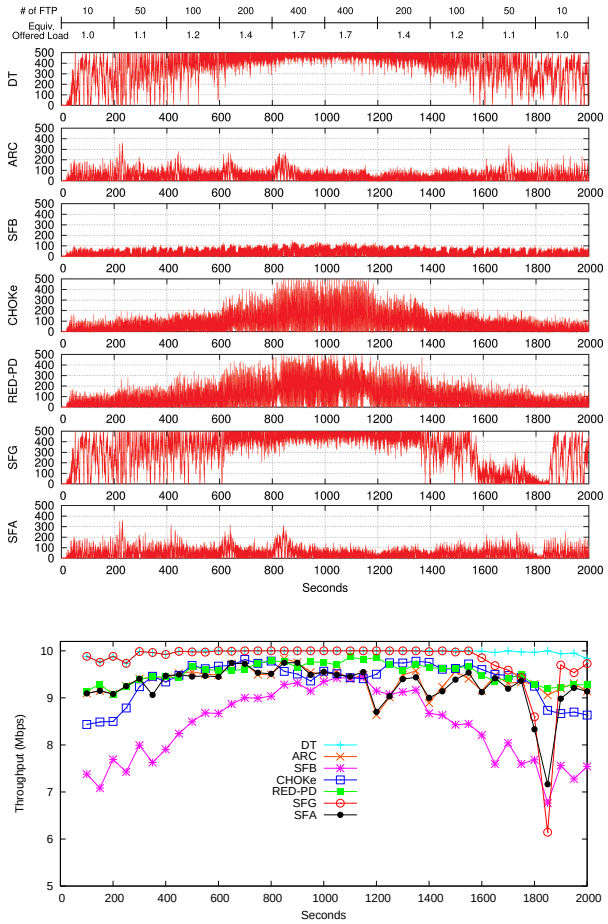

Figure 4: TCP Traffic Mix - Queue Dynamics (top) and Throughput (bottom)

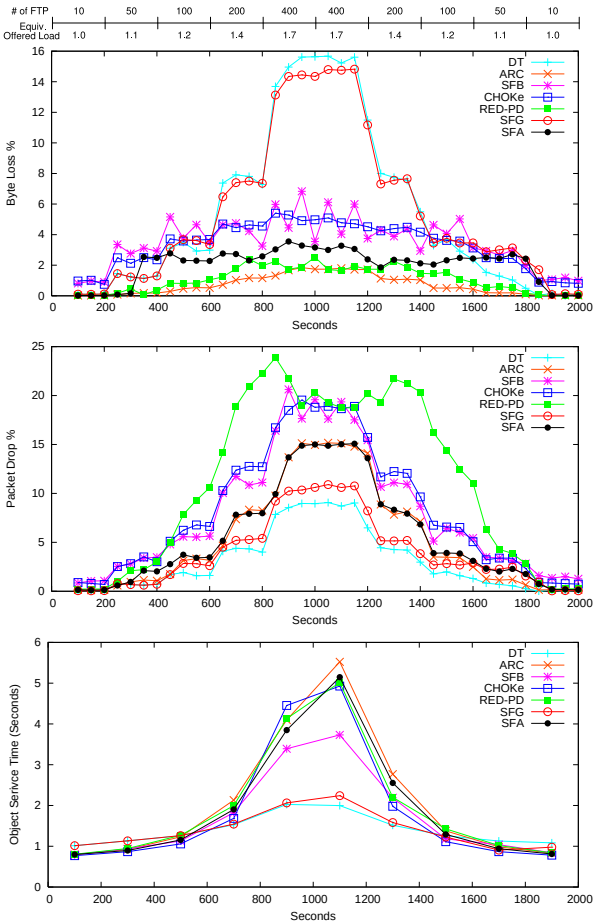

Figure 5: TCP Traffic Mix - Packet Drop Rate (top), Byte Loss Rate (middle) and Average Web Object Service Time (bottom)

video VBR flows.

5.1 TCP Traffic Mix

As a baseline, this experiment compares the performance of the various statistical preferential drop mechanisms with that of a Drop-Tail (DT) and ARC 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, ARC, SFB, CHOKe, RED-PD, SFG and SFA. 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 ARC shows the potential benefits of using AQM: control over link quality of service (QoS) (low queue length) and efficient link utilization. ARC 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. ARC 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 [2].

The packet loss rate and the average Web object service time of Drop-Tail and ARC 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 ARC, the packet losses for ARC, 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, ARC, 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, ARC, 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 ARC 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 ARC 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 [22], 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 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 SFA closely matches that of ARC except for the slightly higher byte loss rates, indicating SFG interfered little with the ability of ARC to control TCP traffic. The sharp link utilization drops for SFG and SFA between 1800 and 1900 seconds are due to the maximum allowed per-flow rate of SFG being less than the TCP-Friendly rate as the system load goes down. However, both SFG and SFA detect the decrease in load within a minute and then turn off the fairness enforcement mechanism.

Comparing the performance of SFB, CHOKe and RED-PD with that of SFA in Figure 4 and Figure 5, SFB, CHOKe and RED-PD have consistently higher packet drop rates and byte loss rates than SFA, 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, was not able to benefit from this higher RED queue length due to its rather inefficient statistical preferential dropping mechanism, and had a low average throughput of 8.5 Mbps with the low offered loads during the first and last 200 seconds. Throughout the simulation, SFB suffers from low link utilization caused by the inefficient rate control afforded by Blue for the traffic mix.

5.2 An Unresponsive, High-Bitrate Flow

In this set of simulations, one unresponsive 10 Mbps CBR UDP flow is added to the TCP traffic mix used in Section 5.1, starting at time 100 seconds and stopping at time 1700 seconds in order to test the performance of the different preferential dropping mechanisms. For comparison, the performance of Drop-Tail and ARC are examined to determine the impact of the unimpeded CBR flow. Figure 6 shows the queue dynamics, Figure 7 shows the system throughput (top) and the throughput of the single CBR stream (bottom), and Figure 8 shows the average Web object service time of the systems.

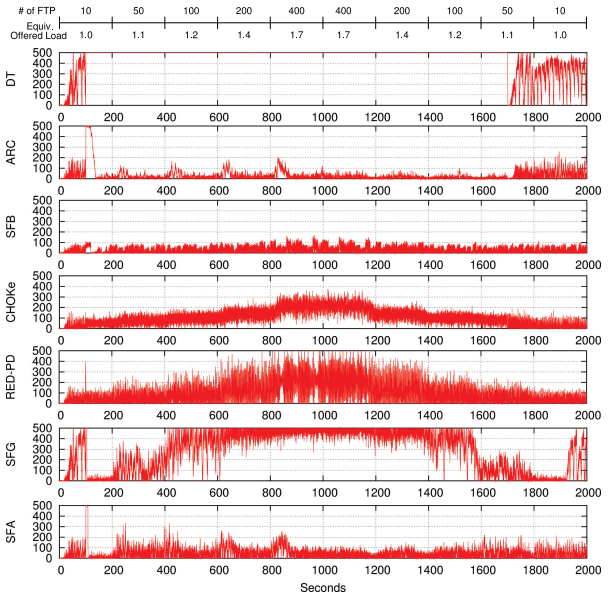

Figure 6: An Unresponsive High-Bitrate CBR Flow - Queue Dynamics

Figure 6 shows that the Drop-Tail queue remains full from the time the high-bitrate CBR flow starts until it stops and Figure 7 (bottom) shows that Drop-Tail is very unfair as about 95% of the link capacity is used by the CBR flow. The average Web service time for Drop-Tail ranges from about 50 to 300 seconds, too high to be seen in Figure 8.

ARC controls the aggregated traffic and the unresponsive flow better than Drop-Tail, applying a high congestion notification probability (CNP) to drop the UDP packets while marking the ECN-enabled packets. Furthermore, as shown in Figure 6, ARC is still able to keep the queue length consistently low while maintaining a high link utilization even in the supersaturated conditions. However, like Drop-Tail, ARC is unfair, and the Web traffic experiences high service times, ranging from about 2 to 13 seconds throughout the simulation.

Figure 7 (bottom) shows that SFB is able to effectively handle the high-bitrate CBR flow, reducing its achieved bandwidth to the target rate. Yet, as in the case of the TCP-only traffic mix, SFB experiences se-

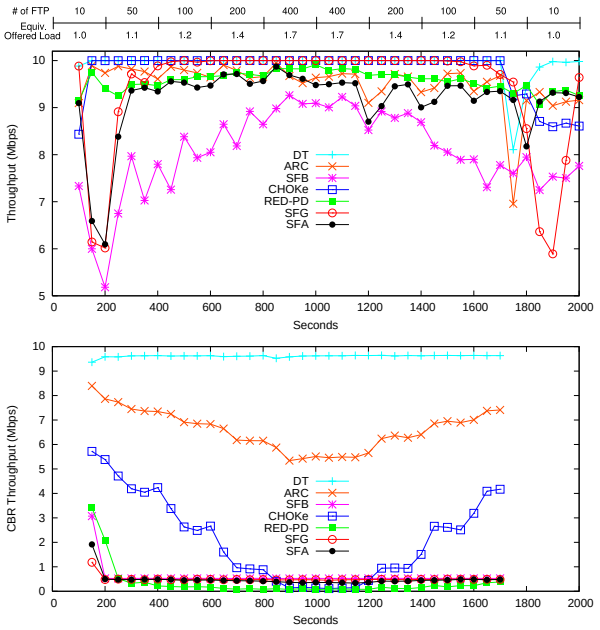

Figure 7: An Unresponsive High-Bitrate CBR Flow - System Throughput (top) and CBR Throughput (bottom)

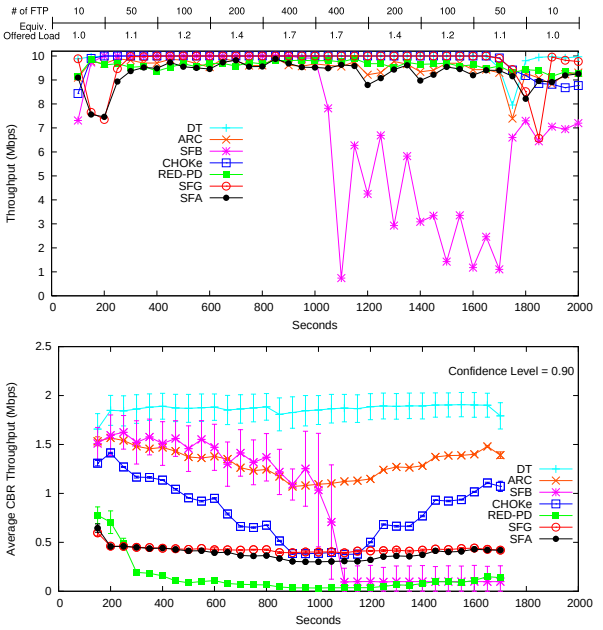

Figure 9: Multiple Unresponsive Medium-Bitrate CBR Flows - System Throughput (top) and CBR Throughput (bottom)

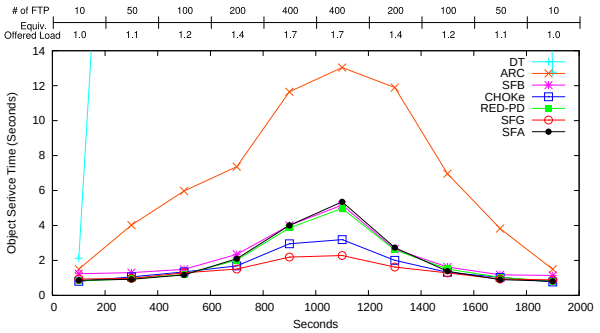

Figure 8: An Unresponsive High-Bitrate CBR Flow - Average Web Object Service Time

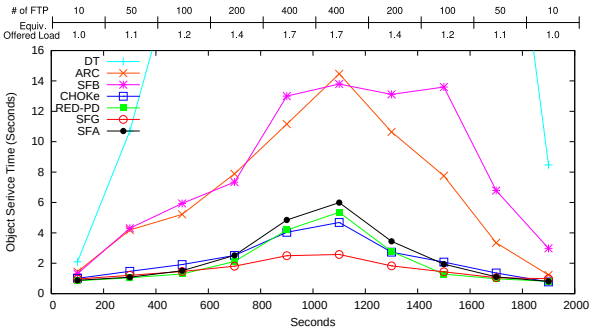

Figure 10: Multiple Unresponsive Medium-Bitrate CBR Flows - Average Web Object Service Time

vere link underutilization. CHOKe, using its statistical filtering mechanism, is able to regulate the unresponsive, high-bitrate flow. However, CHOKe's fairness is coarse as 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 flow by monitoring and then restricting the flow to no more than the periodically adjusted TCP-friendly rate. As observed from the queue dynamics of both the RED AQM based CHOKe and RED-PD, by frequently adjusting the maximum allowed flow rate for the CBR flow, RED-PD affects the stability of the RED congestion control, causing the queue in RED-PD to oscillate more than the queue in CHOKe.

Similarly to SFB and RED-PD, both SFG (with DropTail) and SFA (SFG + ARC) are able to effectively restrict the rate of the unresponsive high-bitrate CBR flow to the target maximum. Yet, Figure 7 (top) shows that a high-bitrate, unresponsive flow in lightly loaded traffic conditions can degrade link utilization by forcing a pre-configured target rate to be imposed on all incoming flows. This potential shortcoming can be relaxed by dynamically adjusting the configuration of N, as briefly discussed in Section 4. Finally, the consistently stable and low queue dynamics in Figure 4 (top) of Section 5.1 and Figure 6 show that the statistical filtering mechanism of SFG does not noticeably affects the congestion control of ARC.

5.3 Multiple Unresponsive, Medium-Bitrate Flows

For the simulation in this section, the one 10 Mbps unresponsive CBR flows used in Section 5.2 is replaced with five unresponsive 2 Mbps CBR flows. As in the previous simulation, the unresponsive CBR flows are started at 100 seconds and stopped at 1700 seconds. Figure 9 shows the system throughput (top) and the average throughput of the five unresponsive CBR streams (bottom), and Figure 10 shows the average Web object service times. To save space, the queue dynamics are not shown, since they are very similar to the queue dynamics in Figure 6 in Section 5.2 and Figure 11 in the next section.

The effect of five unresponsive 2 Mbps CBR flows on the performance of Drop-Tail and ARC is similar to that of a single unresponsive 10 Mbps CBR flows. However, the five smaller capacity flows cause a remarkable degradation in performance for SFB. Figure 9 (bottom) shows SFB fails to detect the unresponsive medium bitrate flows until the offered load reaches 1.7, and performs even more unfairly than ARC, despite ARC not having any fairness protection mechanisms. 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 10, that is similar to or larger than that of ARC for the second half of the simulation.

Other preferential dropping mechanisms perform as designed. RED-PD effectively regulates each unresponsive flow to the estimated TCP-friendly rate at each control epoch, and SFG and SFA prevent each flow from using more bandwidth than the pre-assigned target rate of 0.5 Mbps.

5.4 Multiple Unresponsive, MPEG Video Streams

In order to test SFB, CHOKe, RED-PD, SFG and SFA with more realistic unresponsive flows, the five unresponsive 2 Mbps CBR flows used in Section 5.3 are replaced with five unresponsive MPEG-like video streams, implemented using tools from [10]. The MPEG streams have average I-, P- and B-frame sizes of 11, 8, and 2 Kbytes, respectively based on a trace of typical MPEG-1 video, sent at 30 frames per second for an average bitrate of slightly over 1 Mbps.

As in the previous experiments, the five MPEG streams are started at 100 seconds to stopped at 1700 seconds. For completeness, this section includes the same set of performance results as Section 5.2. Figure 11 shows the queue dynamics of the different systems, Figure 12 shows the system throughput (top) and the average throughput of the five MPEG streams (bottom), and Figure 13 shows the average Web object service times.

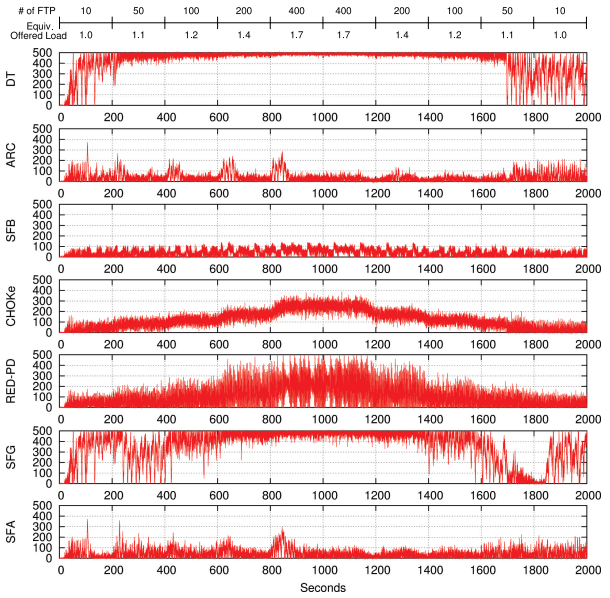

Figure 11: Multiple Unresponsive MPEG Video Streams - Queue Dynamics

As in the case with five unresponsive 2 Mbps CBR flows, for the MPEG streams, all preferential dropping mechanisms perform well, except SFB which again fails to detect the unresponsive flows. Figure 11 and Figure 12 (top) show SFA performs best in terms of queue length, stability and control of link utilization. Considering the Web performance in Figure 13, at an offered load of 1.2 or less, all preferential drop AQM mechanisms outperform Drop-Tail and ARC. When the offered load goes beyond 1.2, all ECN-based mechanisms perform worse than Drop-Tail, where SFG with Drop-Tail queue performs best.

From Figure 12 (top), after 1700 seconds, the SFG and SFA's turning on/off mechanism in response to load is more accurate when used with ARC than with Drop-Tail. The turn on/off decisions are based on the estimated congestion notification rate (CNR) of the system which can be explicitly informed by ARC, whereas for the Drop-Tail queue, SFG has to measure the packet overflow rate to estimate the CNR.

In terms of fairness enforcement in Figure 12 (bottom), RED-PD, with its heavy-weight TCP-friendly target flow rate computation, perform best followed by SFG and SFA. For some additional complexity, the fair-

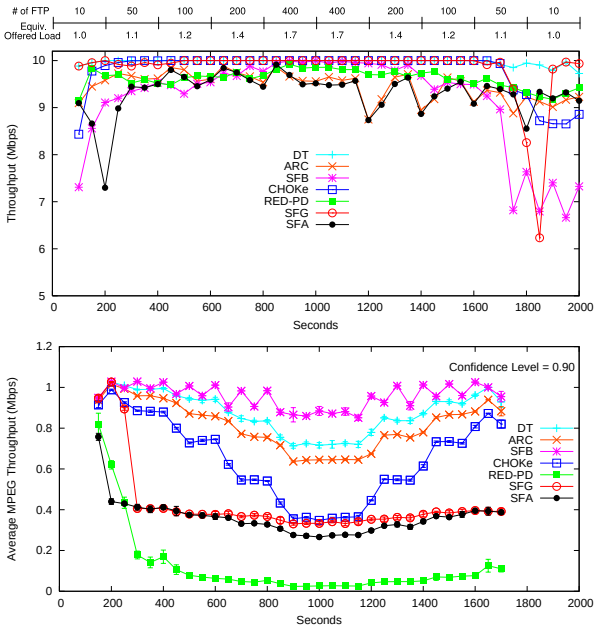

Figure 12: Multiple Unresponsive MPEG Video Streams - System Throughput and MPEG Throughput

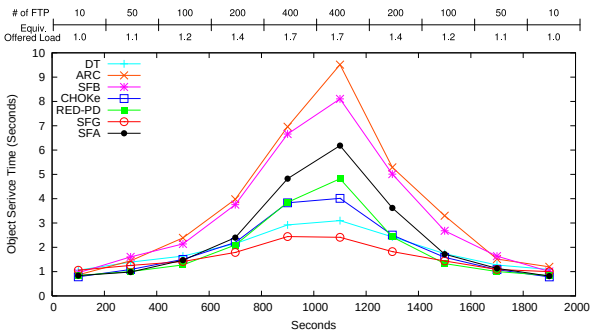

Figure 13: Multiple Unresponsive MPEG Video Streams - Average Web Object Service Time

ness control performance gap between RED-PD and SFG and SFA could be reduced by dynamically adjusting the target flow rate, as briefly discussed in Section 4 (and left as future work). However, even without this adjustment, the main goal of SFG and SFA has been met, and that is not to strictly enforce TCP-Friendly fairness but rather provide reasonable protection from egregiously unresponsive flows.

6 Related Work

This section provides details on the most pertinent related work: Stochastic Fair Blue (SFB) [13], RED with Preferential Dropping (RED-PD) [24] and CHOKe [27] (which stands for CHOose and Keep/Kill).

SFB, RED-PD and CHOKe provide approximate fairness among individual flows without requiring per-flow state information or an edge-core DiffServ [5] network architecture. These mechanisms address the scalability issues of per-flow traffic management by using statistical flow-monitoring or packet-filtering techniques that detect or filter outstanding flows by keeping limited per-flow information. RED-PD requires the most per-flow state information, SFB the second most, and CHOKe the least. Although these mechanisms cannot guarantee total per-flow fairness, they can be scalably deployed independent of other changes at any router with only moderate overhead. Moreover, whether total per-flow fairness is necessary, or even beneficial for flows with heterogenous round-trip times, is arguable.

SFB uses a Bloom filter to offer both a fair congestion notification service and a flow monitoring-based fairness protection service. SFB assigns incoming flows into different groups (accounting bins) multiple times and maintains the congestion notification probability (CNP) for each flow group using the BLUE AQM [12]. SFB determines the CNP for an individual flow by taking the minimum CNP of the groups to which the flow belongs. To determine if a specific flow is consuming more bandwidth than other flows, SFB monitors the maximum CNP of the bins to which a flow belongs. When the maximum CNP of the flow is greater than or equal to a given threshold (0.98 is recommended value), the flow is put in to a penalty box and gets rate limited for a fixed amount of time.

RED-PD is a statistical flow monitoring mechanism that extends RED to support pseudo per-flow bandwidth fairness by recording the per-flow packet drop history to detect potentially high-bandwidth flows during congestion. Although the drop history-based method for selecting flows to monitor reduces the amount of state information required, the flow selection algorithm is complex and requires a significant overhead due to dynamic the history list lookup and maintenence. Moreover, the flow selection performance is sensitive to the history collection epoch length that is computed based on a TCP-friendly rate formula [15]. Once a flow is selected, RED-PD monitors its flow rate and regulates it under the estimated TCP-friendly rate given by the TCP-friendly formula using the RED CNP and a hard-coded round-trip time as parameters.

CHOKe is a lightweight statistical packet filtering mechanism. For each incoming packet, CHOKe randomly choose a packet from the outbound queue and drops both packets if they are from the same flow. This algorithm is derived from the observation that the higher a flow's bitrate, the more the chance for a packet from the flow to be found in the outbound queue. However, the basic CHOKe algorithm cannot effectively control the target flow rate for unresponsive flows as offered

traffic load changes. Thus, CHOKe extends the number of the random packet matches per incoming packet as offered load increases. To estimate offered load, CHOKe suggests using RED's average queue length. The extended CHOKe algorithm divides RED's minimum and maximum queue thresholds range into m even subregions and applies $2i + 1$ drop comparisons for an incoming packet, where $i = \{0, 1, 2, 3, ..., m - 1\}$ is the subregion ID. Although simple in practice, CHOKe can have significant overhead that increases with the offered load. Additionally CHOKe may not work well with other AQMs, such as ARC, that minimize queue length, since there are not as many packets to randomly compare against as in a RED queue.

7 Summary and Future Work

This paper presents a novel statistical packet filtering mechanism, Stochastic Fairness Guardian (SFG), which protects flows that respond to congestion from unresponsive flows through preferential packet drops before the router queue manager. SFG is a general packet filter that can be used in congestion with a Drop-Tail queue or with an AQM to improve efficiency as well as 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 the false positive rate analysis.

In evaluation, SFG is integrated with a Drop-Tail queue and with a previously developed AQM, called ARC, forming Stochastic Fair ARC (SFA). SFG and SFA are evaluated through simulations and compared with other preferential drop mechanisms including SFB [13], CHOKe [27] and RED-PD [24], and also compared with approaches with no filtering mechanisms including ARC and Drop-Tail. Performance metrics include queue dynamics, throughput, fairness, byte loss rate, packet drop rate and Web object service time.

Considering overall performance and design complexity, SFA outperforms other preferential dropping mechanisms as well as Drop-Tail and ARC over a wide, practical range of traffic loads. AQMs using ECN, while improving Web performance under light or moderate congestion, can severely degrade small Web object service times versus Drop-Tail for offered loads of 1.4 or higher. In such a high traffic load, SFG with Drop-Tail queue management shows the best Web performance.

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 SFA into the Linux kernel and measure the filtering overhead.

References

[1] Anonymous, "Aggregate Rate Control for Efficient and Practical Congestion Managment," Work under submission, 2004.

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

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

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

[5] 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.

[6] B. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, July 1970.

[7] 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.

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

[9] 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.

[10] J. Chung and M. Claypool, "Better-Behaved, Better-Performing Multimedia Networking," in *Proceedings of SCS Euromedia*, May 2000.

[11] 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.

[12] 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 Systems Support for Digital Audio and Video (NOSSDAV)*, June 2001.

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

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

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

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

[24] 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.

[25] 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.

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

[27] 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.

[28] 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.

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

[30] 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)*, Boston, MA, USA, Oct. 2002, pp. 315 – 327. [Online]. Available: http://www.usenix.org/publications/library/proceedings/osdi02/tech/saro%iu.html

[31] G. Silva, A. Datta, and S. P. Bhattacharyya, "PI Stabilization of First-Order Systems with Time Delay," *Automatica*, Dec. 2001.

[32] 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*, Vancouver, British Columbia, Canada, Sept. 1998, pp. 118 – 130.