Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness

Wu-chang Feng Dilip D. Kandlur Debanjan Saha Kang G. Shin

Abstract—

This paper describes and evaluates *Stochastic Fair Blue*(SFB), a novel technique for enforcing fairness among a large number of flows. SFB scalably detects and rate-limits non-responsive flows through the use of a marking probability derived from the BLUE queue management algorithm and a Bloom filter. Using analysis and simulation, SFB is shown to effectively handle non-responsive flows using an extremely small amount of state information.

I. INTRODUCTION

Up until recently, the Internet has mainly relied on the cooperative nature of TCP congestion control in order to limit packet loss and fairly share network resources. Increasingly, however, new applications are being deployed which do not use TCP congestion control and are not responsive to the congestion signals given by the network. Such applications are potentially dangerous because they drive up the packet loss rates in the network and can eventually cause congestion collapse [8, 13]. In order to address the problem of non-responsive flows, a lot of work has been done to provide routers with mechanisms for protecting against them [2, 9]. The idea behind these approaches is to detect non-responsive flows and to limit their rates so that they do not impact the performance of responsive flows.

This paper describes and evaluates *Stochastic Fair* BLUE (SFB), a novel technique for protecting TCP flows against non-responsive flows using the BLUE algorithm [6]. SFB is highly scalable and enforces fairness using an extremely small amount of state and a small amount of buffer space. SFB is based on two independent algorithms. The first is the BLUE queue management algorithm. This algorithm uses a single marking probability to mark packets (using ECN [14]) in times of congestion. The heavier the congestion is, the higher the marking probability. The second algorithm is based on Bloom filters [1]. This algorithm allows for the unique classification of objects through the use of multiple, independent hash functions. Using Bloom filters, object classification can be done with an extremely small amount of state information.

The rest of the paper is organized as follows. Section II briefly describes the BLUE algorithm and Bloom filters. Using these two techniques, Section III describes and evaluates Stochastic Fair BLUE (SFB), an algorithm which scalably enforce fairness amongst a large number of connections using a small amount of buffer space. Section IV compares SFB to other approaches which have been proposed to enforce fairness amongst connections. Finally, Section V concludes with a discussion of future work.

II. BACKGROUND

SFB is a simple modification of the BLUE algorithm. BLUE is a fundamentally different queue management algorithm which uses a single marking probability to manage congestion. BLUE addresses one of the fundamental problems of current active queue management algorithms in that they rely on queue lengths as an estimator of congestion. While the presence of a persistent queue indicates congestion, its length gives very little information as to the severity of congestion, that is, the number of competing connections sharing the link. In a busy period, a single source transmitting at a rate greater than the bottleneck link capacity can cause a queue to build up just as easily as a large number of sources can. Since the RED algorithm relies on queue lengths, it has an inherent problem in determining the severity of congestion. As a result, RED requires a wide range of parameters to operate correctly under different congestion scenarios. While RED can achieve an ideal operating point, it can only do so when it has a sufficient amount of buffer space and is correctly parameterized [18].

The idea behind BLUE, on the other hand, is to perform queue management based directly on packet loss and link utilization rather than on the instantaneous or average queue lengths. BLUE maintains a single probability, which it uses to mark (or drop) packets when they are enqueued. If the queue is continually dropping packets due to buffer overflow, BLUE increments the probability, thus increasing the rate at which it sends back congestion notification. Conversely, if the queue becomes empty or if the link is idle, BLUE decreases its marking probability. This effectively allows BLUE to "learn" the correct rate it needs to send back congestion notification. BLUE has been shown to effectively manage congestion using an extremely small amount of buffer space [6].

SFB is an application of a Bloom filter to the BLUE algorithm. Bloom filters are commonly used in word processing software applications as an efficient means to do spell-checking. They are also used in web caches to efficiently determine the existence of an object in the cache. The idea behind Bloom filters is to use L levels of bins with each level containing N bins. For each level, an independent hash function is used to hash a particular object (URL string, English word, TCP/IP connection ID, etc.) into one of the N bins. Each object is then classified and identified by the bins it maps into in each level. Since an object can map into N possible values at each level, an object is identified by an L-tuple of numbers which range from 1 to N. This effectively gives the algorithm N^L unique "buckets" using $L * N$ number of bins. One application of this filter is in spell checkers. Using sufficiently large values of L and N, the entire English dictionary is run through the Bloom filter. Each bin in this filter has a single bit. For each word in the dictionary, every bin that the word hashes into has its bit set to 1. When a document is then spell-checked, words which do not map into bins that are all 1 are flagged as incorrect.

III. STOCHASTIC FAIR BLUE

A. The algorithm

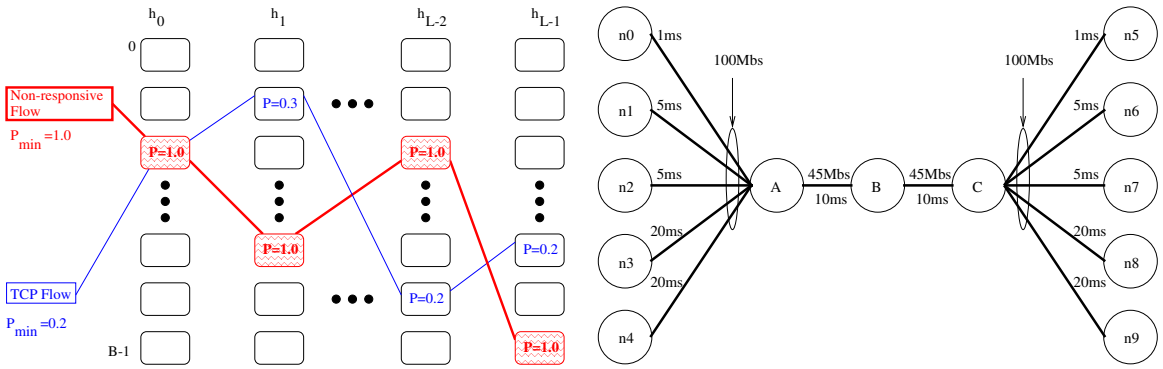
SFB combines BLUE and Bloom filters to produce a highly scalable means to enforce fairness amongst flows using an extremely small amount of state and a small amount of buffer space. Figure 1 shows the basic algorithm. SFB is a FIFO queueing algorithm that identifies and rate-limits non-responsive flows

```
B[l][n]: L x N array of bins
enque()
    Calculate hashes h_0, h_1, ..., h_{L-1};
    Update bins at each level
    for i = 0 to L - 1
        if (B[i][h_i].qlen > bin_size)
            B[i][h_i].p_m += delta;
            Drop packet;
        else if (B[i][h_i].qlen == 0)
            B[i][h_i].p_m -= delta;
    p_min = min(B[0][h_0].p_m .. B[L][h_L].p_m);
    if (p_min == 1)
        ratelimit()
    else
        Mark/drop with probability p_min;
```

Fig. 1. SFB algorithm

based on accounting mechanisms similar to those used with BLUE. SFB maintains $N \times L$ accounting bins. The bins are organized in L levels with N bins in each level. SFB also maintains (L) independent hash functions, each associated with one level of the accounting bins. Each hash function maps a flow into one of the N accounting bins in that level. The accounting bins are used to keep track of queue occupancy statistics of packets belonging to a particular bin. This is in contrast to Stochastic Fair Queueing [11] (SFQ) where the hash function maps flows into separate queues. Each bin in SFB keeps a marking/dropping probability p_m as in BLUE, which is updated based on bin occupancy. As a packet arrives at the queue, it is hashed into one of the N bins in each of the L levels. If the number of packets mapped to a bin goes above a certain threshold (i.e., the size of the bin), p_m for the bin is increased. If the number of packets drops to zero, p_m is decreased.

The observation which drives SFB is that a non-responsive flow quickly drives p_m to 1 in all of the L bins it is hashed into. Responsive flows may share one or two bins with non-responsive flows, however, unless the number of non-responsive flows is



(a) Example of SFB (b) Network topology for evaluation

Fig. 2. SFB example and topology

extremely large compared to the number of bins, a responsive flow is likely to be hashed into at least one bin that is not polluted with non-responsive flows and thus has a normal p_m value. The decision to mark a packet is based on p_{min}, the minimum p_m value of all bins to which the flow is mapped into. If p_{min} is 1, the packet is identified as belonging to a non-responsive flow and is then rate-limited. Note that this approach is akin to applying a Bloom filter on the incoming flows. In this case, the dictionary of messages or words is learned on the fly and consists of the IP headers of the non-responsive flows which are multiplexed across the link. When a non-responsive flow is identified using these techniques, a number of options are available to limit the transmission rate of the flow. In this paper, flows identified as being non-responsive are simply limited to a fixed amount of bandwidth. This policy is enforced by limiting the rate of packet enqueues for flows with p_{min} values of 1. Figure 2(a) shows an example of how SFB works. As the figure shows, a non-responsive flow drives up the marking probabilities of all of the bins it is mapped into. While the TCP flow shown in the figure may map into the same bin as the non-responsive flow at a particular level, it maps into normal bins at other levels. Because of this, the minimum marking probability of the TCP flow is below 1.0 and thus, it is not identified as being non-responsive. On the other hand, since the minimum marking probability of the non-responsive flow is 1.0, it is identified as being non-responsive and rate-limited.

Note that just as BLUE's marking probability can be used in SFB to provide protection against non-responsive flows, it is also possible to apply Adaptive RED's max_p parameter to do the same [5]. In this case, a per-bin max_p value is kept and updated according to the behavior of flows which map into the bin. As with RED, however, there are two problems which make this approach ineffective. The first is the fact that a large amount of buffer space is required in order to get RED to perform well. The second is that the performance of a RED-based scheme is limited since even a moderate amount of congestion requires a max_p setting of 1. Thus, RED, used in this manner, has an extremely difficult time distinguishing between a non-responsive flow and moderate levels of congestion. In order to compare approaches, Stochastic Fair RED (SFRED) was also implemented by applying a Bloom filter to RED.

B. Evaluation

Using `ns`, the SFB algorithm was simulated in the same network as in Figure 2(b) with the transmission delay of all of the links set to $10ms$. The SFB queue is configured with $200KB$ of buffer space and maintains two hash functions each mapping to 23 bins. The size of each bin is set to 13, approximately 50% more than $\frac{1}{23}^{rd}$ of the available buffer space. Note that by allocating more than $\frac{1}{23}^{rd}$ the buffer space to each bin, SFB effectively "overbooks" the buffer in an attempt to improve statistical multiplexing. Notice that even with overbooking, the size of each bin is quite small. Since BLUE performs extremely well under constrained memory resources, SFB can still effectively maximize network efficiency. The queue is also configured to rate-limit non-responsive flows to $0.16Mbs$.

Packet Loss (Mbs)	$2Mbs$ flow			
	SFB	RED	SFRED	SFQ
Total	1.86	1.79	3.10	3.60
$2Mbs$ flow	1.85	0.03	0.63	1.03
All TCP flows	0.01	1.76	2.57	2.47

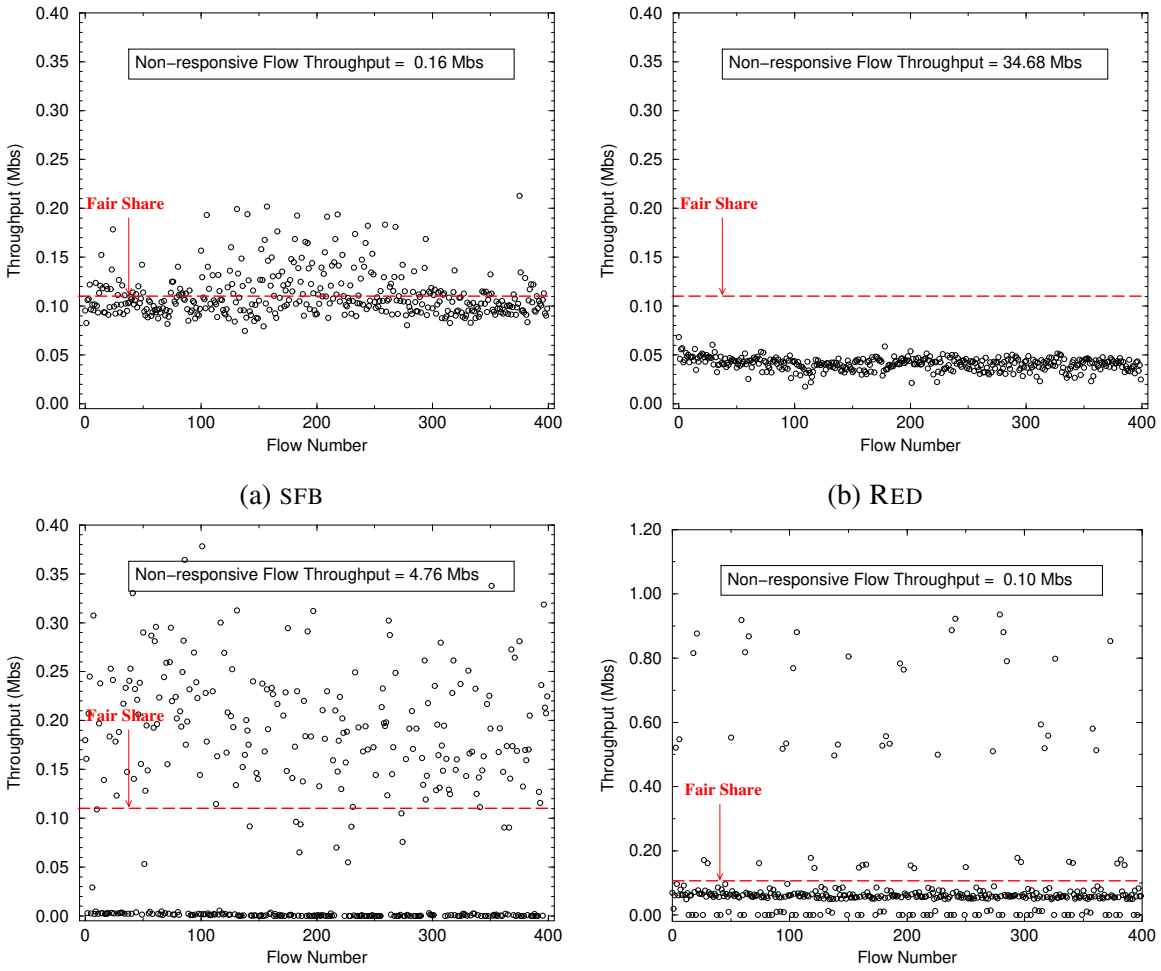
Packet Loss (Mbs)	$45Mbs$ flow			
	SFB	RED	SFRED	SFQ
Total	44.85	13.39	42.80	46.47
$45Mbs$ flow	44.84	10.32	40.24	43.94
All TCP flows	0.01	3.07	2.56	2.53

TABLE I

SFB LOSS RATES (ONE NON-RESPONSIVE FLOW)

In the experiments, 400 TCP sources and one non-responsive, constant rate source are run for 100 seconds from randomly selected nodes in $(n_0, n_1, n_2, n_3, n_4)$ to randomly selected nodes in $(n_5, n_6, n_7, n_8, n_9)$. In one experiment, the non-responsive flow transmits at a rate of $2Mbs$ while in the other, it transmits at a rate of $45Mbs$. Table I shows the packet loss observed in both experiments for SFB. As the table shows, for both experiments, SFB performs extremely well. The non-responsive flow sees almost all of the packet loss as it is rate-limited to a fixed amount of the link bandwidth. In addition, the table shows that in both cases, a very small amount of packets from TCP flows are lost. Table I also shows the performance of RED. In contrast to SFB, RED allows the non-responsive flow to maintain a throughput relatively close to its original sending rate. As a result, the remaining TCP sources see a considerable amount of packet loss which causes their performance to deteriorate. SFRED, on the other hand, does slightly better at limiting the rate of the non-responsive flow, however, it cannot fully protect the TCP sources from packet loss since it has a difficult time discerning non-responsive flows from moderate levels of congestion. Finally, the experiments were repeated using SFQ with an equivalent number of bins (i.e., 46 distinct queues) and a buffer more than twice the size ($414KB$), making each queue equally sized at $9KB$. For each bin in the SFQ, the RED algorithm was applied with min_{th} and max_{th} values set at $2KB$ and $8KB$, respectively. As the table shows, SFQ with RED does an adequate job of protecting TCP flows from the non-responsive flow. However, in this case, partitioning the buffers into such small sizes causes a significant amount of packet loss to occur. Additional experiments show that as the amount of buffer space is decreased even further, the problem is exacerbated and the amount of packet loss increases considerably.

To qualitatively examine the impact that the non-responsive flow has on TCP performance, Figure 3(a) plots the throughput of all 400 TCP flows using SFB when the non-responsive flow sends at a $45Mbs$ rate. As the figure shows, SFB allows each TCP flow to maintain close to a fair share of the bottleneck link's bandwidth while the non-responsive flow is rate-limited to well below its transmission rate. In contrast, Figure 3(b) shows the same experiment using normal RED queue management. The figure shows that the throughput of all TCP flows suffers considerably as the non-responsive flow is allowed to grab a large fraction of the bottleneck link bandwidth. Figure 3(c) shows that while SFRED does succeed in rate-limiting the non-responsive flow, it also manages to drop a significant amount of packets from TCP flows. This is due to the fact that the lack of buffer space and the ineffectiveness of max_p combine to cause SFRED to perform poorly as described in Section III-A. Finally, Figure 3(d) shows that while SFQ with RED can effectively rate-limit the non-responsive flows, the partitioning of buffer space causes the fairness between flows to deteriorate as well. The large amount of packet loss can induce retransmission timeouts across a subset of flows which causes significant amounts of unfairness [12]. Thus, through the course of the experiment, a few TCP flows grab a disproportionate amount of the bandwidth while many of the flows receive significantly less than a fair share of the bandwidth across the link. In addition to this, SFQ with RED allows $\frac{1}{46}^{th}$ of the 400 flows to be mapped into the same queue as the non-responsive flow. Flows that are unlucky enough to map into this bin receive an extremely small amount of the link bandwidth. SFB, in contrast, is able to protect all of the TCP flows in this experiment.



(a) SFB (b) RED

(c) SFRED (d) SFQ+RED

Fig. 3. Bandwidth of TCP flows ($45Mbs$ non-responsive flow)

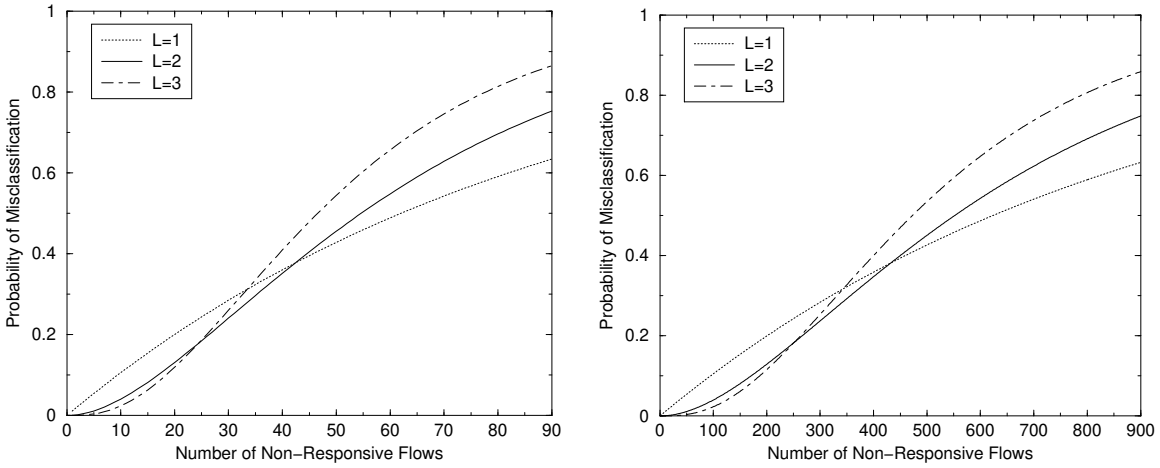
C. *Limitations of* SFB

While it is clear that the basic SFB algorithm can protect TCP-friendly flows from non-responsive flows without maintaining per-flow state, it is important to understand how it works and its limitations. SFB effectively uses L levels with N bins in each level to create N^L virtual buckets. This allows SFB to effectively identify a single non-responsive flow in an N^L flow aggregate using $O(L * N)$ amount of state. For example, in the previous section, using two levels with 23 bins per level effectively creates 529 buckets. Since there are only 400 flows in the experiment, SFB is able to accurately identify and rate-limit a single non-responsive flow without impacting the performance of any of the individual TCP flows. As the number of non-responsive flows increases, the number of bins which become "polluted" or have p_m values of 1 increases. Consequently, the probability that a responsive flow gets hashed into bins which are all polluted, and thus becomes misclassified, increases. Clearly, misclassification limits the ability of SFB to protect well behaved TCP flows.

Using simple probabilistic analysis, Equation (1) gives a closed-form expression of the probability that a well-behaved TCP flow gets misclassified as being non-responsive as a function of number of levels (L), the number of bins per level (B), and the number of non-responsive/malicious flows (M), respectively.

$$p = [1 - (1 - \frac{1}{B})^M]^L \quad (1)$$

In this expression, when L is 1, SFB behaves much like SFQ. The key difference is that SFB using one level is still a FIFO queueing discipline with a shared



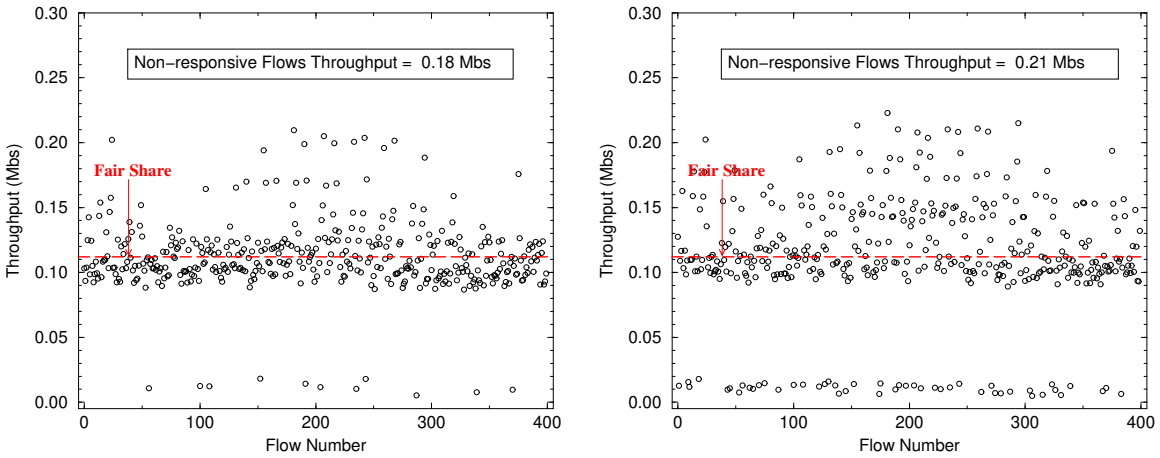
(a) 90 bins (b) 900 bins

Fig. 4. Probability of misclassification

buffer while SFQ has separate per-bin queues and partitions the available buffer space amongst them.

Using the result from Equation (1), it is possible to optimize the performance of SFB given *a priori* information about its operating environment. Suppose the number of simultaneously active non-responsive flows can be estimated (M) and the amount of memory available for use in the SFB algorithm is fixed (C). Then, by minimizing the probability function in Equation (1) with the additional boundary condition that $L \times N = C$, SFB can be tuned for optimal performance. To demonstrate this, the probability for misclassification across a variety of settings is evaluated. Figure 4 shows the probability of misclassifying a flow when the total number of bins is fixed at 90 and 900. In these figures, the number of levels used in SFB along with the number of non-responsive flows are varied. As the figures show, when the number of non-responsive flows is small compared to the number of bins, the use of multiple levels keeps the probability of misclassification extremely low. However, as the number of non-responsive flows increases past half the number of bins present, the single level SFB queue affords the smallest probability of misclassification. This is due to the fact that when the bins are distributed across multiple levels, each non-responsive flow pollutes a larger number of bins. For example, using a single level SFB queue with 90 bins, a single non-responsive flow pollutes only one bin. Using a two-level SFB queue with each level containing 45 bins, the number of effective bins is 45×45 (2025). However, a single non-responsive flow pollutes two bins (one per level). Thus, the advantage gained by the two-level SFB queue is lost when additional non-responsive flows are added, as a larger fraction of bins become polluted compared to the single-level situation.

To examine the performance degradation of SFB as the number of non-responsive flows increases, Figure 5 shows the bandwidth plot of the 400 TCP flows when 4 and 8 non-responsive flows are present. In these experiments, each non-responsive flow transmits at a rate of $5Mbs$. As Equation (1) predicts, in an SFB configuration that contains two levels of 23 bins, 2.65% of the TCP flows (11) are misclassified when 4 non-responsive flows are present. Similarly, when 8 non-responsive flows are present, 8.96% (36) of the TCP flows are misclassified. When the number of non-responsive flows approaches N, the performance of SFB deteriorates quickly as an increasing number of bins at each level becomes polluted. In the case of 8 non-responsive flows, approximately 6 bins or one-fourth of the bins in each level are polluted. As the figure shows, the number of misclassified flows matches the model quite closely. Note that even though a larger number of flows are misclassified as the number of non-responsive flows increases, the probability of misclassification in a two-level SFB still remains below that of SFQ or a single-level SFB. Using the same number of bins (46), the equation predicts that SFQ and a single-level SFB misclassify 8.42% of the TCP



(a) 4 non-responsive flows (b) 8 non-responsive flows

Fig. 5. Bandwidth of TCP flows using SFB

flows (34) when 4 non-responsive flows are present and 16.12% of the TCP flows (64) when 8 non-responsive are present.

D. SFB *with moving hash functions*

In this section, two basic problems with the SFB algorithm are addressed. The first, as described above, is to mitigate the effects of misclassification. The second is to be able to detect when non-responsive flows become responsive and to reclassify them when they do.

The idea behind SFB with moving hash functions is to periodically or randomly reset the bins and change the hash functions. A non-responsive flow will continually be identified and rate-limited regardless of the hash function used. However, by changing the hash function, responsive TCP flows that happen to map into polluted bins will potentially be remapped into at least one unpolluted bin. Note that this technique effectively creates virtual bins across time just as the multiple levels of bins in the original algorithm creates virtual bins across space. In many ways the effect of using moving hash functions is analogous to channel hopping in CDMA [17] systems. It essentially reduces the likelihood of a responsive connection being continually penalized due to erroneous assignment into polluted bins.

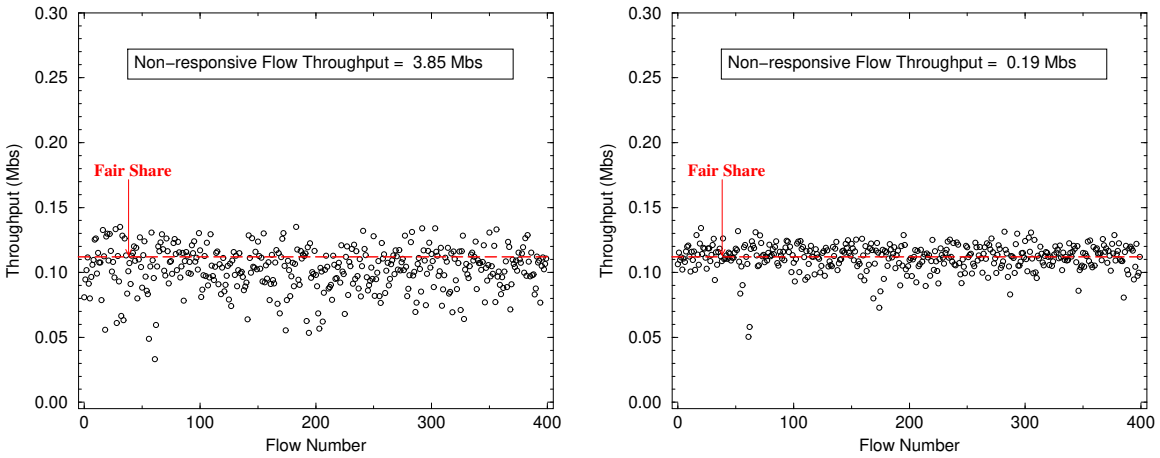
To show the effectiveness of this approach, the idea of moving hash functions was applied to the experiment in Figure 5(b). In this experiment, 8 non-responsive flows along with 400 responsive flows share the bottleneck link. To protect against continual misclassification, the hash function is changed every two seconds. Figure 6(a) shows the bandwidth plot of the experiment. As the figure shows, SFB performs fairly well. While flows are sometimes misclassified causing a degradation in performance, none of the TCP-friendly flows are shut out due to misclassification. This is in contrast to Figure 5 where a significant number of TCP flows receive very little bandwidth.

Packet Loss (Mbs)	10-30s	30-50s	50-70s
TCP Flows	0.402	0.358	0.260
Non-responsive	4.866	4.849	4.898
Oscillating	4.871	0.025	4.845
Total	10.139	5.232	10.003

TABLE II

SFB LOSS RATES (OSCILLATING FLOW EXPERIMENT)

While the moving hash functions improve fairness across flows in the experiment, it is interesting to note that every time the hash function is changed and the bins are reset, non-responsive flows are temporarily placed on “parole”. That is, non-responsive flows are given the benefit of the doubt and are no longer rate-limited. Only after these flows cause sustained packet loss, are they identified and rate-limited again. Unfortunately, this can potentially allow such flows to grab much more than their fair share of bandwidth over time. For example, as Fig-



(a) Moving hash (b) Double buffered moving hash

Fig. 6. Bandwidth of TCP flows using modified SFB algorithms

ure 6(a) shows, non-responsive flows are allowed to consume $3.85Mbs$ of the bottleneck link. One way to solve this problem is to use two sets of bins. As one set of bins is being used for queue management, a second set of bins using the next set of hash functions can be warmed up. In this case, any time a flow is classified as non-responsive, it is hashed using the second set of hash functions and the marking probabilities of the corresponding bins in the warmup set are updated. When the hashes are switched, the bins which have been warmed up are then used. Thus, non-responsive flows are rate-limited right from the beginning. Figure 6(b) shows the performance of the double buffered moving hash. The algorithm effectively controls the bandwidth of the non-responsive flows and affords the TCP flows a very high level of protection.

One of the advantages of the moving hash function is that it can quickly react to non-responsive flows which become TCP-friendly. In this case, changing the hash bins places the newly reformed flow out on parole for good behavior. Only after the flow resumes transmitting at a high rate, is it again rate-limited. To show this, an additional experiment was run using the same experimental setup as above. In this experiment, one non-responsive flow with a transmission rate of $5Mbs$ and one oscillating flow is run between network endpoints. The oscillating flow transmits at $5Mbs$ from $t = 10s$ to $t = 30s$ and from $t = 50s$ to $t = 70s$. At all other times, the flow transmits at $0.10Mbs$, approximately a fair share of the bottleneck link. Table II shows the packet loss rates in the experiment. As the table shows, the first non-responsive flow sees a sustained packet loss rate throughout the experiment which effectively limits its throughput to well below its transmission rate. The table also shows that when the second flow transmits at $5Mbs$, it observes a sustained packet loss rate as a large fraction of its packets are dropped by the queue. When the second flow cuts its transmission rate to a fair share of the link's bandwidth, it is reclassified and a very small fraction of its packets are dropped. Finally, the table shows that all 400 TCP flows see a minimal amount of packet loss throughout the experiment. Figure 7 shows the bandwidth plot for the TCP flows in the experiment. As shown in the figure, SFB protects the TCP flows from the non-responsive flows, thus allowing them to maintain close to a fair share of the bottleneck link.

E. Round-trip time sensitivity

The previous experiments with SFB use a network topology in which all of the connections have approximately the same round-trip time. When a large number of connections with varying round-trip times are used with SFB, fairness between flows can deteriorate. It has been shown that TCP connections with smaller round-trip times can dominate the bandwidth on the bottleneck link since their window increases are clocked more frequently. When a small number of such connections are present, SFB can mitigate this problem somewhat. Similar to the non-responsive flow cases above, TCP connections with small round-trips slowly drive the marking proba-

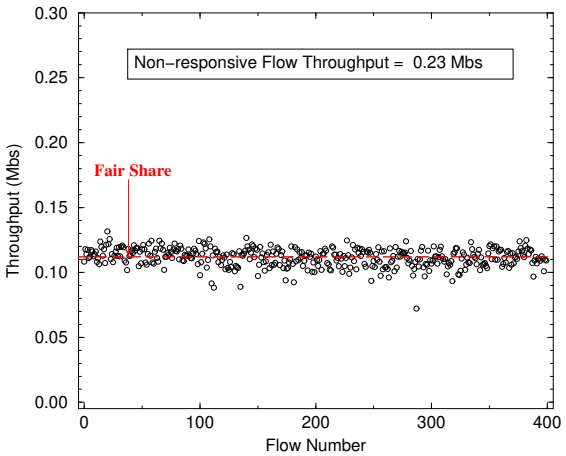


Fig. 7. Bandwidth of TCP flows (One non-responsive, one oscillating flow)

bility of their bins higher. Thus, when p_{min} is calculated, they receive a larger fraction of congestion notification. However, when a large number of TCP flows with varying round-trip times are present, this mechanism breaks down just as SFB breaks down with a large number of non-responsive flows.

Figure 8 shows the performance of RED and SFB using the original network shown in Figure 2(b). Using this network, 400 sources are randomly started between network end points. As the figure shows, both RED and SFB show biases towards connections with smaller round-trip times. However, since all of the flows still use TCP, the amount of unfairness between flows is limited.

IV. Comparison to Other Approaches

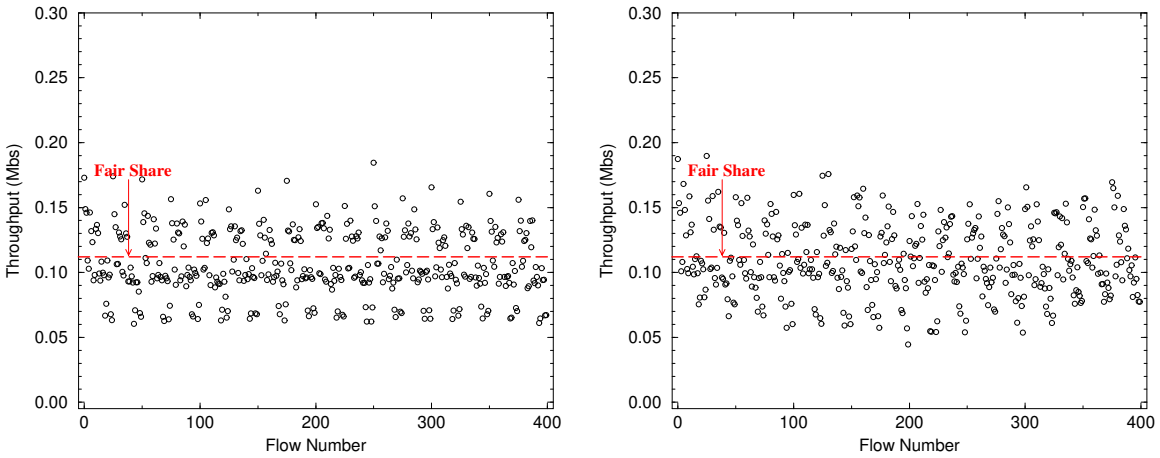
SFB provides one particular solution for identifying and rate-limiting non-responsive flows, thereby enforcing fairness. This section compares SFB to other related approaches.

A. RED *with Penalty Box*

The RED with penalty box approach uses a finite log of recent packet loss events. The algorithm identifies flows which are non-responsive based on the log [10] and takes corrective action. Flows which are identified as being non-responsive are rate-limited using a mechanism such as class-based queueing [7]. While this approach may be viable under certain circumstances, it is unclear how the algorithm performs in the face of a large number of non-responsive flows. Unless the packet loss log is large, a single set of high bandwidth flows can dominate the loss log and allow other, non-responsive flows to go through without rate-limitation. In addition, flows which are classified as non-responsive remain in the "penalty box" even if they subsequently become responsive to congestion. A periodic and explicit check is required to move flows out of the penalty box. Finally, the algorithm relies on a TCP-friendliness check in order to determine whether or not a flow is non-responsive. Without *a priori* knowledge of the round-trip time of every flow being multiplexed across the link, it is difficult to accurately determine whether or not a connection is TCP-friendly.

B. FRED

Another proposal for using RED mechanisms to provide fairness is Flow-RED (FRED) [9]. The idea behind FRED is to keep state based on instantaneous queue occupancy of a given flow. If a flow continually occupies a large amount of the queue's buffer space, it is detected and limited to a smaller amount of the buffer space. While this scheme provides rough fairness in many situations, since the algorithm only keeps state for flows which have packets queued at the bottleneck link, it requires a large amount of buffers to work well. Without sufficient buffer space, it becomes hard for FRED to detect non-responsive flows since they may not have enough packets continually queued to trigger the detection mechanism. In addition, non-responsive flows are immediately re-classified as being responsive as soon as they clear their packets from the congested queue. For small queue sizes, it is quite easy to construct a transmission pattern which circumvents FRED's protection mechanisms. Note that SFB



(a) RED (b) SFB

Fig. 8. Bandwidth of TCP flows over varying round-trip times.

does not directly rely on queue occupancy statistics, but rather long-term packet loss and link utilization behavior. Because of this, SFB is better suited for protecting TCP flows against non-responsive flows using a minimal amount of buffer space. Finally, as with the packet loss log approach, FRED also has a problem when dealing with a large number of non-responsive flows. In this situation, the ability to distinguish these flows from normal TCP flows deteriorates considerably since the queue occupancy statistics used in the algorithm become polluted. By not using packet loss as a means for identifying non-responsive flows, FRED cannot make the distinction between N TCP flows multiplexed across a link versus N non-responsive flows multiplexed across a link.

C. RED *with per-flow Queueing*

A RED-based, per-active flow approach has also been proposed for providing fairness [16]. The idea behind this approach is to do per-flow accounting and queueing only for flows which are active. The approach argues that since keeping a large amount of state is feasible, per-flow queueing and accounting is possible even in the core of the network. The drawbacks of this approach is that it provides no savings in the amount of state required. If N flows are active, $O(N)$ amount of state must be kept to isolate the flows from each other. In addition, this approach does not address the large amount of legacy hardware which exists in the network. For such hardware, it may be infeasible to provide per-flow queueing and accounting. Because SFB provides considerable savings in the amount of state and buffers required, it is a more viable alternative.

D. *Stochastic Fair Queueing*

Stochastic Fair Queueing (SFQ) is similar to an SFB queue with only one level of bins. The biggest difference is that instead of having separate queues, SFB uses the hash function for accounting purposes. Thus, SFB has two fundamental advantages over SFQ. The first is that it can make better use of its buffers. SFB gets some statistical multiplexing of buffer space as it is possible for the algorithm to overbook buffer space to individual bins in order to keep the buffer space fully utilized. As described in Section III-B, partitioning the available buffer space adversely impacts the packet loss rates and the fairness amongst TCP flows. The other key advantage is that SFB is a FIFO queueing discipline. As a result, it is possible to change the hash function on the fly without having to worry about packet re-ordering caused by mapping of flows into a different set of bins. Without additional tagging and book-keeping, applying the moving hash functions to SFQ can cause significant packet re-ordering.

E. *Core-Stateless Fair Queueing*

Core-Stateless Fair Queueing [15] (CSFQ) is a highly scalable approach for enforcing fairness between flows without keeping any state in the core of the network. The approach relies on per-flow accounting and marking at the edge of the network in conjunction with a probabilistic dropping mech-

anism in the core of the network. The idea behind CSFQ is to estimate the rate of the flow at the ingress of the network or network cloud and to attach an estimate of the flow's sending rate to *every* packet that the flow sends. Given this label, intermediate routers at congested links in the network calculate a dropping probability which is derived from an estimate of a fair share of the bottleneck link capacity and the rate of the flow as identified in the label.

While CSFQ provides an elegant and efficient solution to providing fairness, it relies on the use of additional information that is carried in every packet of the flow. Thus, the scheme trades off overhead in the packet header at every network link for resource management overhead at the bottleneck router. In addition, it requires that both intermediate routers and edge devices adhere to the same labeling and dropping algorithm. A misconfigured or poorly implemented edge device can significantly impact the fairness of the scheme. SFB, on the other hand, does not rely on coordination between intermediate routers and edge markers and can peform well without placing additional overhead in packet headers.

V. Conclusion and Future Work

This paper has demonstrated the efficacy of SFB a new queue management algorithm for protetcting TCP flows against non-responsive flows. As part of on-going work, several extensions to SFB are being considered. In particular, additional mechanisms for managing non-responsive flows are being examined. In this paper, non-responsive flows were rate-limited to a fixed amount of bandwidth across the bottleneck link. However, it is possible to rate-limit non-responsive flows to a fair share of the link's capacity. One way to do this is to estimate both the number of non-responsive flows and the total number of flows going through the bottleneck. Using this information, the rate-limiting mechanism can be set accordingly. Another possible mechanism to find the number of "polluted" bins and use it to derive the fraction of flows which are non-responsive. Assuming perfect hash functions, this can be directly derived from simple analytical models of SFB as described in Section III. Finally, the development of an "enhanced" BLUE queue management algorithm which is similar to "enhanced" RED [3, 4] is being considered. By using BLUE, the buffer requirements needed to support differentiated services can be greatly reduced.

References

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

[2] K. Fall and S. Floyd. Router Mechanisms to Support End-to-End Congestion Control. ftp://ftp.ee.lbl.gov/papers/collapse.ps, February 1997.

[3] W. Feng, D. Kandlur, D. Saha, and K. Shin. Understanding TCP Dynamics in an Integrated Services Internet. In *Proc. of NOSSDAV '97*, May 1997.

[4] W. Feng, D. Kandlur, D. Saha, and K. Shin. Adaptive Packet Marking for Providing Differentiated Services in the Internet. In *Proc. of ICNP '98*, October 1998.

[5] W. Feng, D. Kandlur, D. Saha, and K. Shin. A Self-Configuring RED Gateway. In *Proc. IEEE INFOCOM*, March 1999.

[6] W. Feng, D. Kandlur, D. Saha, and K. Shin. Blue: A New Class of Active Queue Management Algorithms. In *UM CSE-TR-387-99*, April 1999.

[7] S. Floyd and V. Jacobson. Link-sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transactions on Networking*, 3(4), August 1995.

[8] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of ACM SIGCOMM*, pages 314–329, August 1988.

[9] D. Lin and R. Morris. Dynamics of Random Early Detection. In *Proc. of ACM SIGCOMM*, September 1997.

[10] S. McCanne and S. Floyd. http://www-nrg.ee.lbl.gov/ns/. ns-LBNL Network Simulator, 1996.

[11] P. McKenney. Stochastic Fairness Queueing. In *Proc. IEEE INFOCOM*, March 1990.

[12] R. Morris. TCP Behavior with Many Flows. In *Proc. IEEE International Conference on Network Protocols*, October 1997.

[13] V. Paxson. End-to-End Internet Packet Dynamics. In *Proc. of ACM SIGCOMM*, September 1997.

[14] K. Ramakrishnan and S. Floyd. A Proposal to Add Explicit Congestion Notification (ECN) to IP. *RFC 2481*, January 1999.

[15] I. Stoica, S. Shenker, and H. Zhang. Core-Stateless Fair Queueing: A Scalable Architecture to Approximate Fair Bandwidth Allocations in High Speed Networks. In *Proceedings of ACM SIGCOMM*, September 1998.

[16] B. Suter, T. V. Lakshman, D. Stiliadis, and A. Choudhury. Design Considerations for Supporting TCP with Per-flow Queueing. *Proc. IEEE INFOCOM*, March 1998.

[17] V. K. Garg and K. Smolik and J. E. Wilkes. Applications Of CDMA In Wireless/Personal Communications. Prentice Hall Professional Technical Reference, October 1996.

[18] C. Villamizar and C. Song. High Performance TCP in ANSNET. *Computer Communication Review*, 24(5):45–60, October 1994.