BLUE: A New Class of Active Queue Management Algorithms

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

†Department of EECS
University of Michigan
Ann Arbor, MI 48105
Phone: (313) 763-5363 Fax: (313) 763-4617
{wuchang,kgshin}@eecs.umich.edu

‡Network Systems Department
IBM T.J. Watson Research Center
Yorktown Heights, NY 10598
Phone: (914) 784-7194 Fax: (914) 784-6205
{kandlur,debanjan}@watson.ibm.com

Abstract

In order to stem the increasing packet loss rates caused by an exponential increase in network traffic, the IETF is considering the deployment of active queue management techniques such as RED [13]. While active queue management can potentially reduce packet loss rates in the Internet, this paper shows that current techniques are ineffective in preventing high loss rates. The inherent problem with these queue management algorithms is that they all use queue lengths as the indicator of the severity of congestion. In light of this observation, a fundamentally different active queue management algorithm called BLUE is proposed. BLUE uses packet loss and link idle events to manage congestion. Using simulation and controlled experiments, BLUE is shown to perform significantly better than RED both in terms of packet loss rates and buffer size requirements in the network. As an extension to BLUE, a novel technique for enforcing fairness among a large number of flows is described. In particular, this paper proposes and evaluates Stochastic Fair BLUE (SFB), a queue management algorithm which can identify and rate-limit non-responsive flows using a very small amount of state information.

Keywords: Congestion control, Internet, TCP, RED, queue management

1 Introduction

It is important to avoid high packet loss rates in the Internet. When a packet is dropped before it reaches its destination, all of the resources it has consumed in transit are wasted. In extreme cases, this situation can lead to congestion collapse [17]. Improving the congestion control and queue management algorithms in the Internet has been one of the most active areas of research in the past few years. While a number of proposed enhancements have made their way into actual implementations, connections still experience high packet loss rates. Loss rates are especially high during times of heavy congestion, when a large number of connections compete for scarce network bandwidth. Recent measurements have shown that the growing demand for network bandwidth has driven loss rates up across various links in the Internet [23]. In order to stem the increasing packet loss rates caused by an exponential increase in network traffic, the IETF is considering the deployment of explicit congestion notification (ECN) [11, 24, 25] along with active queue management techniques such as RED (Random Early Detection) [2, 11]. While ECN is necessary for eliminating packet loss in the Internet [10], this paper shows that RED, even when used in conjunction with ECN, is ineffective in preventing packet loss.

The basic idea behind RED queue management is to detect incipient congestion *early* and to convey congestion notification to the end-hosts, allowing them to reduce their transmission rates before queues in the network overflow and packets are dropped. To do this, RED maintains an exponentially-weighted moving average of the queue length which it uses to detect congestion. When the average queue length exceeds a minimum threshold (min_{th}), packets are randomly dropped or marked with an explicit congestion notification (ECN) bit. When the average queue length exceeds a maximum threshold, all packets are dropped or marked. While RED is certainly an improvement over traditional drop-tail queues, it has several shortcomings. One of the fundamental problems with RED and all other known active queue management techniques is 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 [5, 29].

In light of the above observation, this paper proposes BLUE, a fundamentally different active queue management algorithm which uses packet loss and link utilization history to manage congestion. BLUE maintains a single probability, which it uses to mark (or drop) packets when they are queued. If the queue is continually dropping packets due to buffer overflow, BLUE increments the marking 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. Using simulation and experimentation, this paper demonstrates the superiority of BLUE to RED in reducing packet losses even when operating with a smaller buffer. Using mechanisms based on BLUE, a novel mechanism for effectively and scalably enforcing fairness among a large number of flows is also proposed and evaluated.

The rest of the paper is organized as follows. Section 2 gives a description of RED and shows why it is ineffective at managing congestion. Section 3 describes BLUE and provides a detailed analysis and evaluation of its performance. Section 4 describes and evaluates Stochastic Fair BLUE (SFB), an algorithm based on BLUE which scalably enforces fairness amongst a large number of connections. Section 5 compares SFB to other approaches which have been proposed to enforce fairness amongst connections. Finally, Section 6 concludes with a discussion of future work.

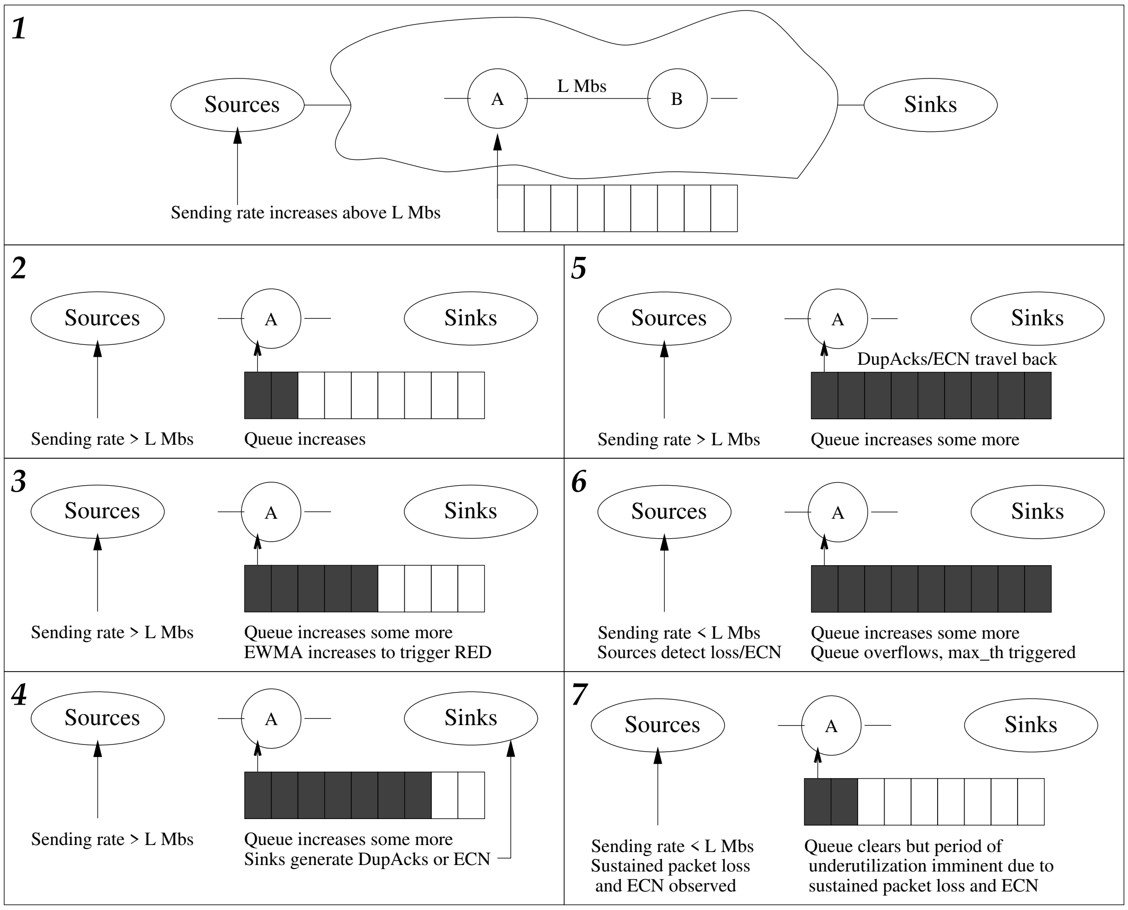


Figure 1: RED example

2 Background

One of the biggest problems with TCP's congestion control algorithm over drop-tail queues is that the sources reduce their transmission rates only after detecting packet loss due to queue overflow. Since considerable amount of time may elapse between the packet drop at the router and its detection at the source, a large number of packets may be dropped as the senders continue transmission at a rate that the network cannot support. RED alleviates this problem by detecting incipient congestion *early* and delivering congestion notification to the end-hosts, allowing them to reduce their transmission rates before queue overflow occurs. In order to be effective, a RED queue must be configured with a sufficient amount of buffer space to accommodate an applied load greater than the link capacity from the instant in time that congestion is detected using the queue length trigger to the instant in time that the applied load decreases at the bottleneck link in response to congestion notification. RED also must ensure that congestion notification is given at a rate which sufficiently suppresses the transmitting sources without underutilizing the link. Unfortunately, when a large number of TCP sources are active, the aggregate traffic generated is extremely bursty [7, 10]. Bursty traffic often defeats the active queue management techniques used by RED since queue lengths grow and shrink rapidly, well before RED can react. Figure 1 shows a simplified pictorial example of how RED functions under this congestion scenario.

The congestion scenario presented in Figure 1 occurs when a large number of TCP sources are active

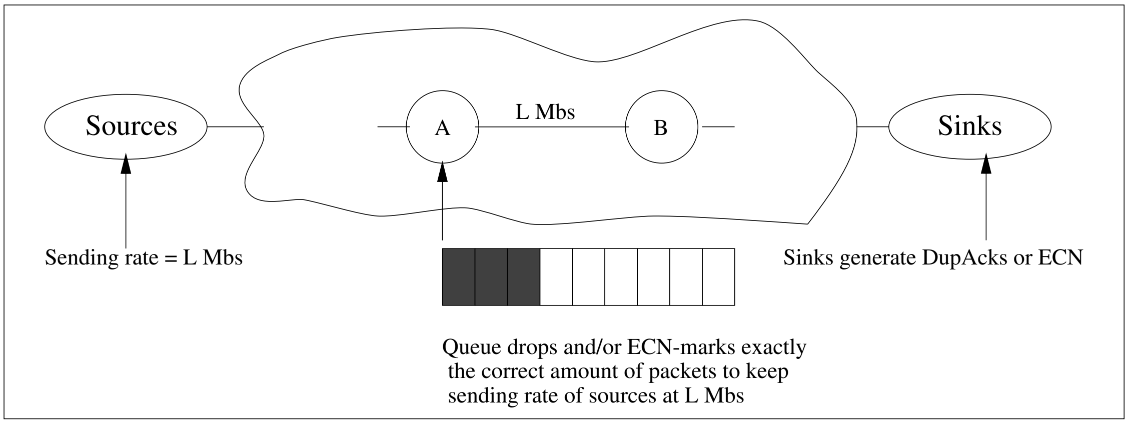


Figure 2: Ideal scenario

and when a small amount of buffer space is used at the bottleneck link. As the figure shows, at $t = 1$, a sufficient change in aggregate TCP load (due to TCP opening its congestion window) causes the transmission rates of the TCP sources to exceed the capacity of the bottleneck link. At $t = 2$, the mismatch between load and capacity causes a queue to build up at the bottleneck. At $t = 3$, the average queue length exceeds min_{th} and the congestion-control mechanisms are triggered. At this point, congestion notification is sent back to the end hosts at a rate dependent on the queue length and marking probability max_p. At $t = 4$, the TCP receivers either detect packet loss or observe packets with their ECN bits set. In response, duplicate acknowlegdements and/or TCP-based ECN signals are sent back to the sources. At $t = 5$, the duplicate acknowlegements and/or ECN signals make their way back to the sources to signal congestion. At $t = 6$, the sources finally detect congestion and adjust their transmission rates. Finally, at $t = 7$, a decrease in offered load at the bottleneck link is observed. Note that it has taken from $t = 1$ until $t = 7$ before the offered load becomes less than the link's capacity. Depending upon the aggressiveness of the aggregate TCP sources [7, 10] and the amount of buffer space available in the bottleneck link, a large amount of packet loss and/or deterministic ECN marking may occur. Such behavior leads to eventual underutilization of the bottleneck link.

One way to solve this problem is to use a large amount of buffer space at the RED gateways. For example, it has been suggested that in order for RED to work well, an intermediate router requires buffer space that amounts to twice the bandwidth-delay product [29]. This approach, in fact, has been taken by an increasingly large number of router vendors. Unfortunately, in networks with large bandwidth-delay products, the use of a large amounts of buffer adds considerable end-to-end delay and delay jitter. This severely impacts the ability to run interactive applications. In addition, the abundance of deployed routers which have limited memory resources makes this solution undesirable.

Figure 2 shows how an ideal queue management algorithm works. In this figure, the congested gateway delivers congestion notification at a rate which keeps the aggregate transmission rates of the TCP sources at or just below the clearing rate. While RED can achieve this ideal operating point, it can do so only when it has a sufficiently large amount of buffer space and is correctly parameterized.

3 BLUE

In order to remedy the shortcomings of RED, this section proposes and evaluates a fundamentally different queue management algorithm called BLUE. Using both simulation and experimentation, BLUE is shown to

```
Upon packet loss (or Q_len > L) event:
    if ( (now - last_update) > freeze_time ) then
        p_m = p_m + d1
        last_update = now
Upon link idle event:
    if ( (now - last_update) > freeze_time ) then
        p_m = p_m - d2
        last_update = now
```

Figure 3: The BLUE algorithm

overcome many of RED's shortcomings. RED has been designed with the objective to (1) minimize packet loss and queueing delay, (2) avoid global synchronization of sources, (3) maintain high link utilization, and (4) remove biases against bursty sources. This section shows how BLUE either improves or matches RED's performance in all of these aspects. The results also show that BLUE converges to the ideal operating point shown in Figure 2 in almost all scenarios, even when used with very small buffers.

3.1 The algorithm

The key idea behind BLUE is to perform queue management based directly on packet loss and link utilization rather than on the instantaneous or average queue lengths. This is in contrast to all known active queue management schemes which use some form of queue occupancy in their congestion management. BLUE maintains a single probability, p_m, 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 p_m, 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. Figure 3 shows the BLUE algorithm. Note that the figure also shows a variation to the algorithm in which the marking probability is updated when the queue length exceeds a certain value. This modification allows room to be left in the queue for transient bursts and allows the queue to control queuing delay when the size of the queue being used is large. Besides the marking probability, BLUE uses two other parameters which control how quickly the marking probability changes over time. The first is *freeze_time*. This parameter determines the minimum time interval between two successive updates of p_m. This allows the changes in the marking probability to take effect before the value is updated again. While the experiments in this chapter fix *freeze_time* as a constant, this value should be randomized in order to avoid global synchronization [12]. The other parameters used, ($d1$ and $d2$), determine the amount by which p_m is incremented when the queue overflows or is decremented when the link is idle. For the experiments in this paper, $d1$ is set significantly larger than $d2$. This is because link underutilization can occur when congestion management is either too conservative or too aggressive, but packet loss occurs only when congestion management is too conservative. By weighting heavily against packet loss, BLUE can quickly react to a substantial increase in traffic load. Note that there are a myriad of ways in which p_m can be managed. While the experiments in this paper study a small range of parameter settings, experiments with additional

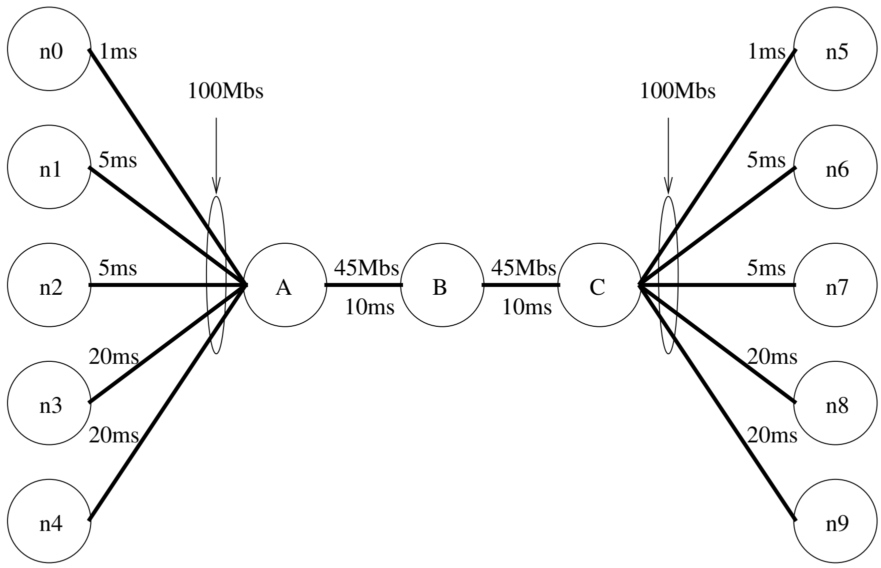


Figure 4: Network topology

Configuration	w_q
$R1$	0.0002
$R2$	0.002
$R3$	0.02
$R4$	0.2

Table 1: RED configurations

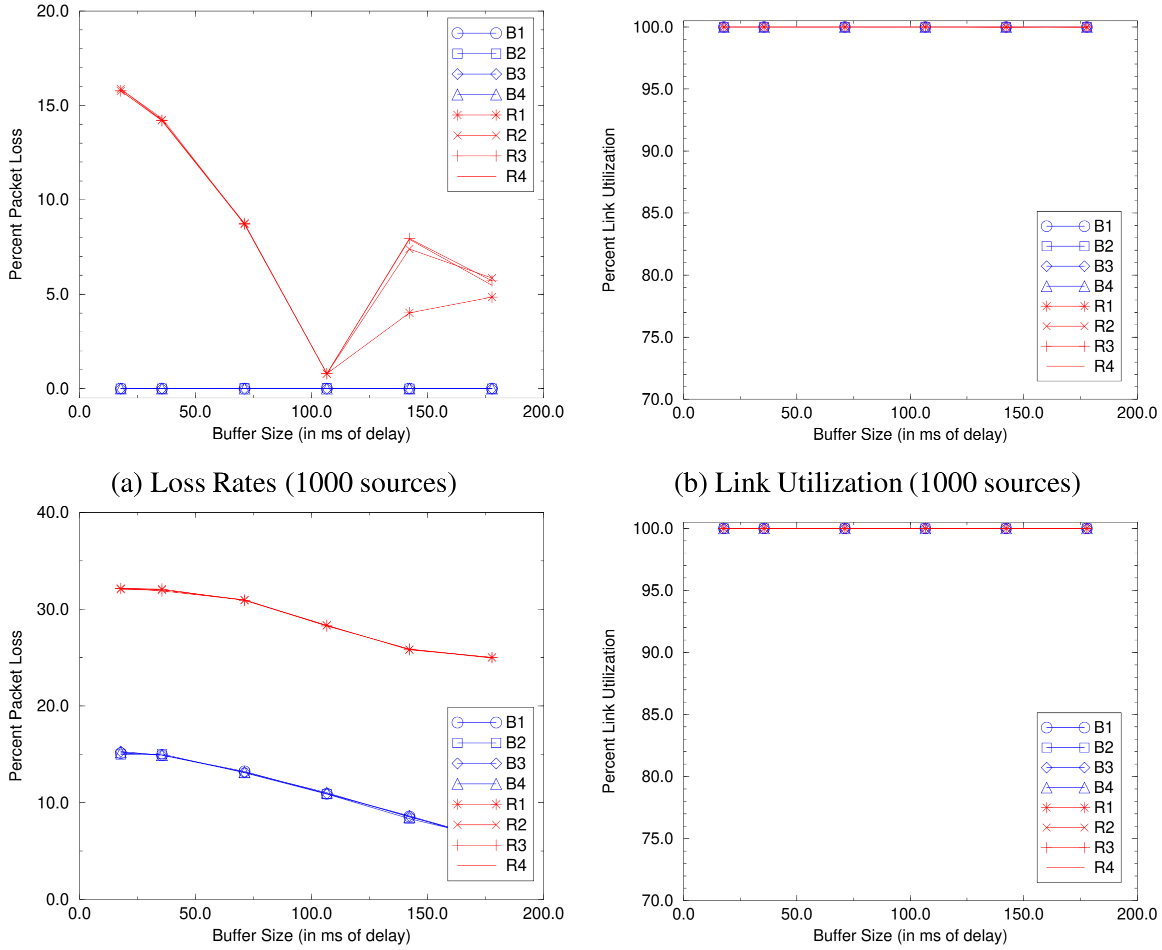
parameter settings and algorithm variations have also been performed with the only difference being how quickly the queue management algorithm adapts to the offered load. While BLUE seems extremely simple, it provides a significant performance improvement even when compared to a RED queue which has been reasonably configured.

3.2 Packet loss rates using RED and BLUE

In order to evaluate the performance of BLUE, a number of experiments were run using `ns` [19] over a small network shown in Figure 4. Using this network, Pareto on/off sources with mean on-times of 2 seconds and mean off-times of 3 seconds were run from one of the leftmost nodes (n_0, n_1, n_2, n_3, n_4) to one of the rightmost nodes (n_5, n_6, n_7, n_8, n_9). In addition, all sources were enabled with ECN support, were randomly started within the first second of simulation, and used $1KB$ packets. Packet loss statistics were then measured after 100 seconds of simulation for 100 seconds. Loss statistics were also measured for RED using the same network and under identical conditions. For the RED queue, min_{th} and max_{th} were set to 20% and 80% of the queue size, respectively. RED's congestion notification mechanism was made as aggressive as possible by setting max_p to 1. For these experiments, this is the ideal setting of max_p since it minimizes both the queueing delay and packet loss rates for RED [10]. Given these settings, a range of RED configurations are studied which vary the value of w_q, the weight in the average queue length calculation

Configuration	$freeze\_time$	$d1$	$d2$
$B1$	$10ms$	0.0025	0.00025
$B2$	$100ms$	0.0025	0.00025
$B3$	$10ms$	0.02	0.002
$B4$	$100ms$	0.02	0.002

Table 2: BLUE configurations



(a) Loss Rates (1000 sources)

(b) Link Utilization (1000 sources)

(c) Loss Rates (4000 sources)

(d) Link Utilization (4000 sources)

Figure 5: Packet loss rates of RED and BLUE

for RED. It is interesting to note that as w_q gets smaller, the impact of queue length on RED's congestion management algorithm gets smaller. For extremely small values of w_q, RED's algorithm becomes decoupled from the queue length and thus acts more like BLUE. Table 1 shows the configurations used for RED. For the BLUE experiments, $d1$ and $d2$ are set so that $d1$ is an order of magnitude larger than $d2$. Using these values, the $freeze\_time$ is then varied between $10ms$ and $100ms$. Additional simulations using a wider range of values were also performed and showed similar results.

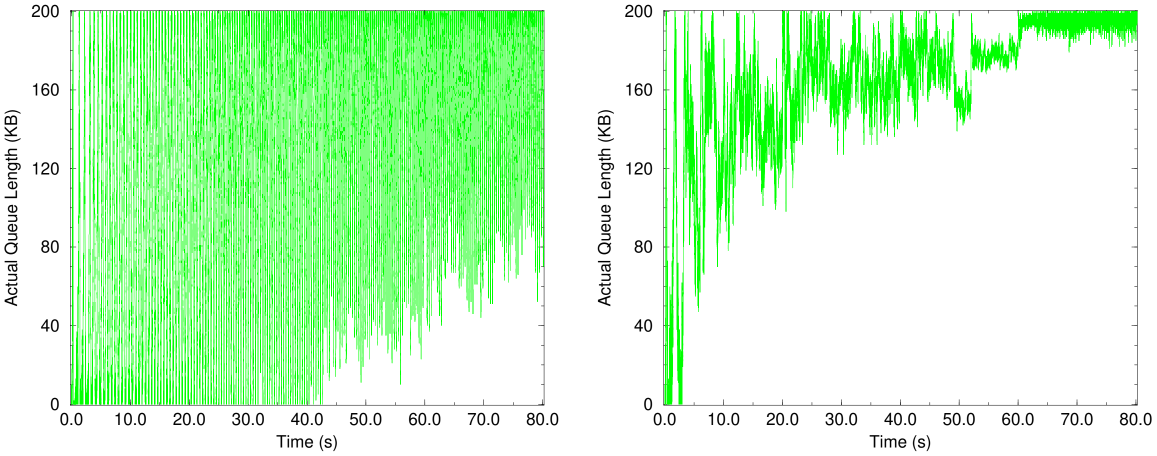
Figure 5 shows the loss rates observed over different queue sizes using both BLUE and RED with 1000 and 4000 connections present. In these experiments, the queue at the bottleneck link between A and B is sized from $100KB$ to $1000KB$. This corresponds to queueing delays which range from $17.8ms$ and $178ms$ as shown in the figure. As Figure 5(a) shows, with 1000 connections, BLUE maintains zero loss rates over all queue sizes even those which are below the bandwidth-delay product of the network [29]. This is in contrast to RED which suffers double-digit loss rates as the amount of buffer space decreases. An interesting point in the RED loss graph shown in Figure 5(a) is that it shows a significant dip in loss rates at a buffering delay of around $80ms$. This occurs because of a special operating point of RED when the average queue length stays above max_{th} all the time. At several points during this particular experiment, the buffering delay and offered load match up perfectly to cause the average queue length to stay at or above max_{th}. In this operating region, the RED queue marks every packet, but the offered load is aggressive enough to keep the queue full. This essentially allows RED to behave at times like BLUE with a marking probability of 1 and a queueing delay equivalent to max_{th}. This unique state of operation is immediately disrupted by any changes in the load or round-trip times, however. When the buffering delay is increased, the corresponding round-trip times increase and cause the aggregate TCP behavior to be less aggressive. Deterministic marking on this less aggressive load causes fluctuations in queue length which can increase packet loss rates since RED undermarks packets at times. When the buffering delay is decreased, the corresponding round-trip times decrease and cause the aggregate TCP behavior to be more aggressive. As a result, packet loss is often accompanied with deterministic marking. When combined, this leads again to fluctuations in queue length. At a load which is perfectly selected, the average queue length of RED can remain at max_{th} and the queue can avoid packet loss and prevent queue fluctuations by marking every packet. Figure 5(b) shows the link utilization across all experiments. As the figure shows, the link remains fully utilized for both RED and BLUE regardless of the parameter settings.

As Figure 5(c) shows, when the number of connections is increased to 4000, BLUE still significantly outperforms RED. Even with an order of magnitude more buffer space, RED still cannot match BLUE's loss rates using $17.8ms$ of buffering at the bottleneck link. It is interesting to note that BLUE's marking probability remains at 1 throughout the duration of all of these experiments. Thus, even though every packet is being marked, the offered load can still cause a significant amount of packet loss. The reason why this is the case is that the TCP sources being used do not invoke a retransmission timeout upon receiving an ECN signal with a congestion window of 1. Section 3.4 shows how this can significantly influence the performance of both RED and BLUE. Figure 5(d) shows the link utilization for all of the 4000 connection experiments. Again, regardless of the parameter settings, both RED and BLUE achieve full link utilization.

The most important consequence of using BLUE is that congestion control can be performed with a minimal amount of buffer space. This reduces the end-to-end delay over the network, which in turn, improves the effectiveness of the congestion control algorithm. In addition, smaller buffering requirements allow more memory to be allocated to high priority packets [4, 15], and frees up memory for other router functions such as storing large routing tables. Finally, BLUE allows legacy routers to perform well even with limited memory resources.

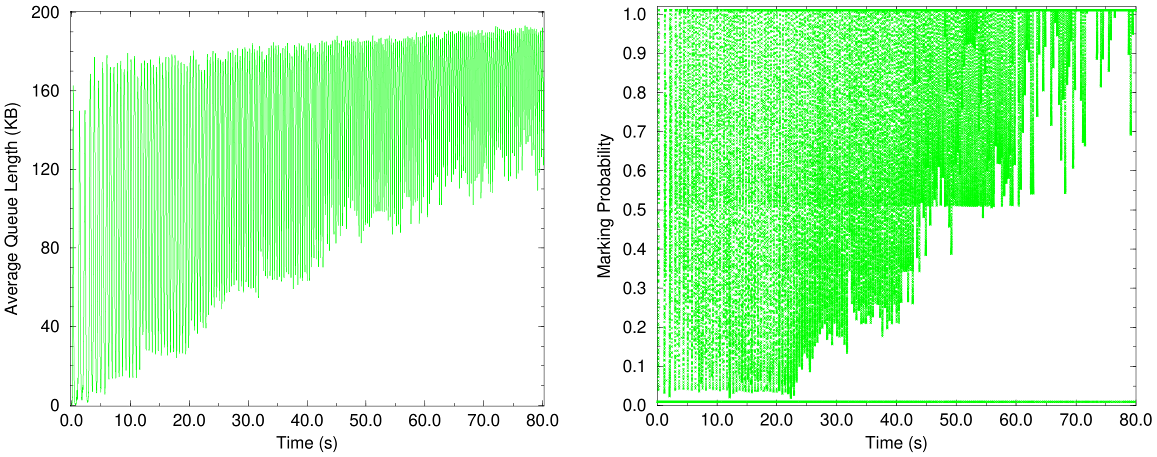
3.3 Understanding BLUE

To fully understand the difference between the RED and BLUE algorithms, Figure 6 compares their queue length plots in an additional experiment using the $B4$ configuration of BLUE and the $R2$ configuration of RED. In this experiment, a workload of infinite sources is changed by increasing the number of connections by 200 every 20 seconds. As Figure 6(a) shows, RED sustains continual packet loss throughout the experiment. In addition, at lower loads, periods of packet loss are often followed by periods of underutilization as deterministic packet marking and dropping eventually causes too many sources to reduce their transmission



(a) RED (b) BLUE

Figure 6: Queue length plots of RED and BLUE



(a) Q_{ave} (b) $\frac{p_b}{1-count \times p_b}$

Figure 7: Marking behavior of RED

rates. In contrast, as Figure 6(b) shows, since BLUE manages its marking rate more intelligently, the queue length plot is more stable. Congestion notification is given at a rate which neither causes periods of sustained packet loss nor periods of continual underutilization. Only when the offered load rises to 800 connections, does BLUE sustain a significant amount of packet loss.

Figure 7 plots the average queue length (Q_{ave}) and the marking probability ($\frac{p_b}{1-count \times p_b}$) of RED throughout the experiment. The average queue length of RED contributes directly to its marking probability since p_b is a linear function of Q_{ave} ($p_b = max_p \times \frac{Q_{ave}-min_{th}}{max_{th}-min_{th}}$). As shown in Figure 7(a), the average queue length of RED fluctuates considerably as it follows the fluctuations of the instantaneous queue length. Because of this, the marking probability of RED, as shown in Figure 7(b), fluctuates considerably as well. In contrast, Figure 8 shows the marking probability of BLUE. As the figure shows, the marking probability converges to a value that results in a rate of congestion notification which prevents packet loss and keeps link utilization high throughout the experiment. In fact, the only situation where BLUE cannot prevent sustained packet

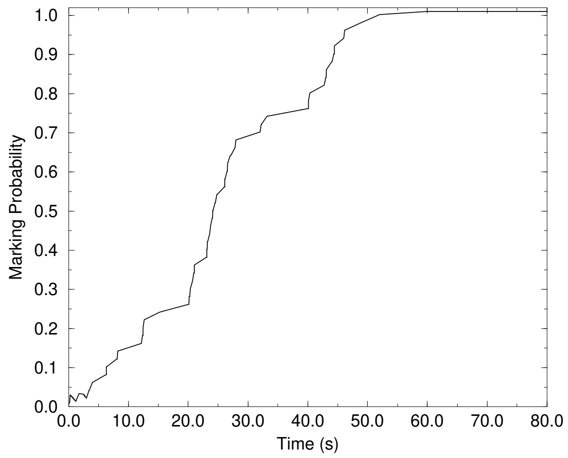
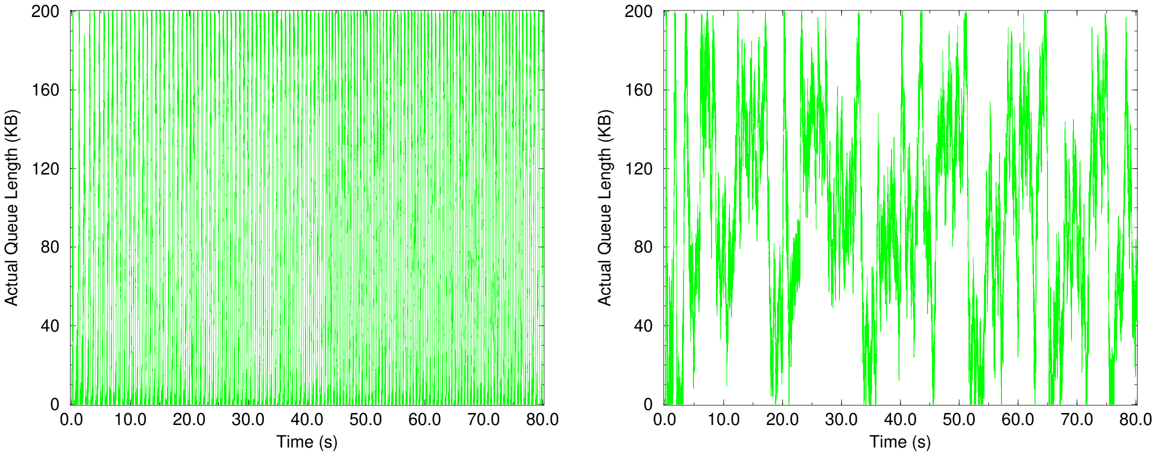


Figure 8: Marking behavior of BLUE (p_m)

loss is when every packet is being marked, but the offered load still overwhelms the bottleneck link. As described earlier, this occurs at $t = 60s$ when the number of sources is increased to 800. The reason why packet loss still occurs when every packet is ECN-marked is that for these sets of experiments, the TCP implementation used does not invoke an RTO when an ECN signal is received with a congestion window of 1. This adversely affects the performance of both RED and BLUE in this experiment. Note that the comparison of marking probabilities between RED and BLUE gives some insight as to how to make RED perform better. By placing a low pass filter on the calculated marking probability of RED, it may be possible for RED's marking mechanism to behave in a manner similar to BLUE's.

While low packet loss rates, low queueing delays, and high link utilization are extremely important, the queue length and marking probability plots allow us to explore the effectiveness of RED and BLUE in preventing global synchronization and in removing biases against bursty sources. RED attempts to avoid global synchronization by randomizing its marking decision and by spacing out its marking. Unfortunately, when aggregate TCP load changes dramatically as it does when a large amount of connections are present, it becomes impossible for RED to achieve this goal. As Figure 7(b) shows, the marking probability of RED changes considerably over very short periods of time. Thus, RED fails to mark packets evenly over time and hence cannot remove synchronization among sources. As Figure 8 shows, the marking probability of BLUE remains steady. As a result, BLUE marks packets randomly and evenly over time. Consequently, it does a better job in avoiding global synchronization.

Another goal of RED is to eliminate biases against bursty sources in the network. This is done by limiting the queue occupancy so that there is always room left in the queue to buffer transient bursts. In addition, the marking function of RED takes into account the last packet marking time in its calculations in order to reduce the probability that consecutive packets belonging to the same burst are marked. Using a single marking probability, BLUE achieves the same goal equally well. As the queue length plot of BLUE shows (Figure 6), the queue occupancy remains below the actual capacity, thus allowing room for a burst of packets. In addition, since the marking probability remains smooth over large time scales, the probability that two consecutive packets from a smoothly transmitting source are marked is the same as with two consecutive packets from a bursty source.



(a) RED

(b) BLUE

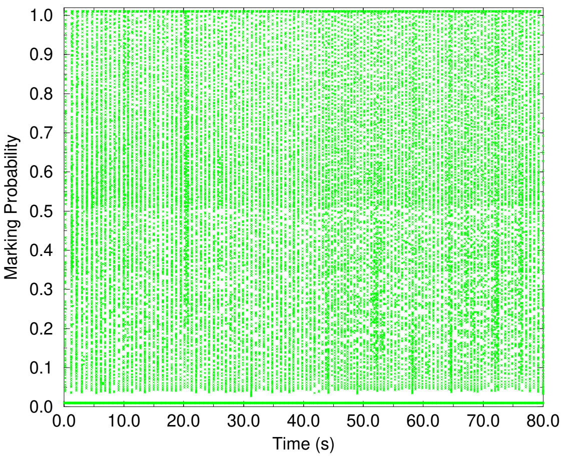
Figure 9: Queue length plots of RED and BLUE with ECN timeouts

3.4 The effect of ECN timeouts

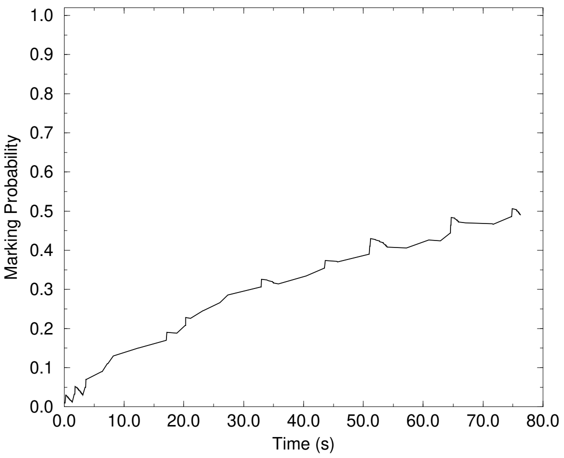
All of the previous experiments use TCP sources which support ECN, but do not perform a retransmission timeout upon receipt of an ECN signal with a congestion window of 1. This has a significant, negative impact on the packet loss rates observed for both RED and BLUE especially at high loads. Figure 9 shows the queue length plot of RED and BLUE using the same experiments as in Section 3.2 with TCP sources enabled with ECN timeouts. Figure 9(a) shows that by deterministically marking packets at max_{th}, RED oscillates between periods of packet loss and periods of underutilization as described in Section 2. Note that this is in contrast to Figure 6(a) where without ECN timeouts, TCP is aggressive enough to keep the queue occupied when the load is sufficiently high. An interesting point to make is that RED can effectively prevent packet loss by setting its max_{th} value sufficiently far below the size of the queue. In this experiment, a small amount of loss occurs since deterministic ECN marking does not happen in time to prevent packet loss. While the use of ECN timeouts allows RED to avoid packet loss, the deterministic marking eventually causes underutilization at the bottleneck link. Figure 9(b) shows the queue length plot of BLUE over the same experiment. In contrast to RED, BLUE avoids deterministic marking and maintains a marking probability that allows it to achieve high link utilization while avoiding sustained packet loss over all workloads.

Figure 10 shows the corresponding marking behavior of both RED and BLUE in the experiment. As the figure shows, BLUE maintains a steady marking rate which changes as the workload is changed. On the other hand, RED's calculated marking probability fluctuates from 0 to 1 throughout the experiment. When the queue is fully occupied, RED overmarks and drops packets causing a subsequent period of underutilization as described in Section 2. Conversely, when the queue is empty, RED undermarks packets causing a subsequent period of high packet loss as the offered load increases well beyond the link's capacity.

Figure 11 shows how ECN timeouts impact the performance of RED and BLUE. The figure shows the loss rates and link utilization using the 1000 and 4000 connection experiments in Section 3.2. As the figure shows, BLUE maintains low packet loss rates and high link utilization across all experiments. The figure also shows that the use of ECN timeouts allows RED to reduce the amount of packet loss in comparison to Figure 5. However, because RED often deterministically marks packets, it suffers from poor link utilization unless correctly parameterized. The figure shows that only an extremely small value of w_q (Configuration $R1$) allows RED to approach the performance of BLUE. As described earlier, a small w_q value effectively decouples congestion management from the queue length calculation making RED queue



(a) $\frac{p_b}{1-count\times p_b}$ of RED



(b) p_m of BLUE

Figure 10: Marking behavior with ECN timeouts

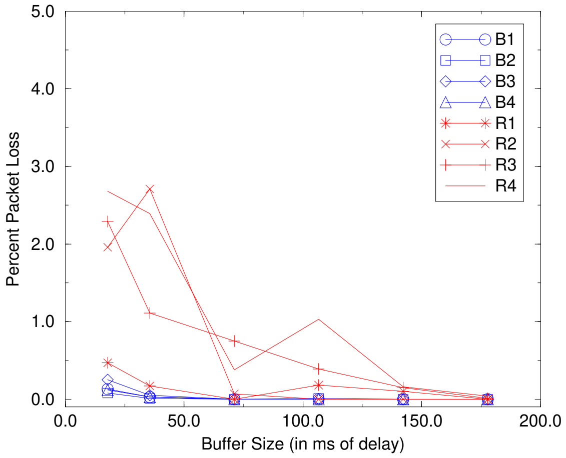
management behave more like BLUE.

3.5 Implementation

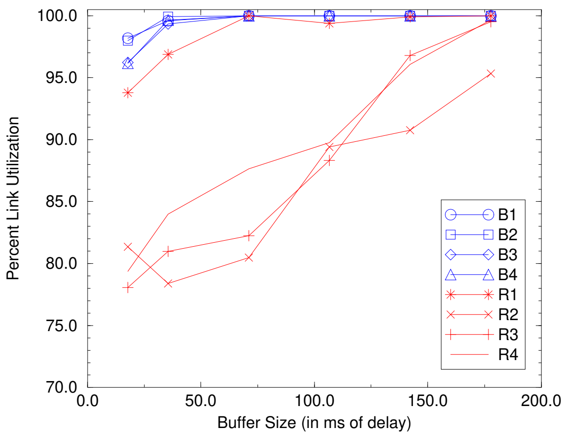
In order to evaluate BLUE in a more realistic setting, it has been implemented in FreeBSD 2.2.7 using ALTQ [3]. In this implementation, ECN uses two bits of the type-of-service (ToS) field in the IP header [25]. When BLUE decides that a packet must be dropped or marked, it examines one of the two bits to determine if the flow is ECN-capable. If it is not ECN-capable, the packet is simply dropped. If the flow is ECN-capable, the other bit is set and used as a signal to the TCP receiver that congestion has occurred. The TCP receiver, upon receiving this signal, modifies the TCP header of the return acknowledgment using a currently unused bit in the TCP flags field. Upon receipt of a TCP segment with this bit set, the TCP sender invokes congestion-control mechanisms as if it had detected a packet loss.

Using this implementation, several experiments were run on the testbed shown in Figure 12. Each network node and link is labeled with the CPU model and link bandwidth, respectively. Note that all links are shared Ethernet segments. Hence, the acknowledgments on the reverse path collide and interfere with data packets on the forward path. As the figure shows, FreeBSD-based routers using either RED or BLUE queue management on their outgoing interfaces are used to connect the Ethernet and Fast Ethernet segments. In order to generate load on the system, a variable number of `netperf` [22] sessions are run from the *IBM PC 360* and the *Winbook XL* to the *IBM PC 365* and the *Thinkpad 770*. The router queue on the congested Ethernet interface of the *Intellistation Zpro* is sized at $50KB$ which corresponds to a queueing delay of about $40ms$. For the experiments with RED, a configuration with a min_{th} of $10KB$, a max_{th} of $40KB$, a max_p of 1, and a w_q of 0.002 was used. For the experiments with BLUE, a $d1$ of 0.01, a $d2$ of 0.001 and a $freeze\_time$ of $50ms$ was used. To ensure that the queue management modifications did not create a bottleneck in the router, the testbed was reconfigured exclusively with Fast Ethernet segments and a number of experiments between network endpoints were run using the BLUE modifications on the intermediate routers. In all of the experiments, the sustained throughput was always above $80Mbs$.

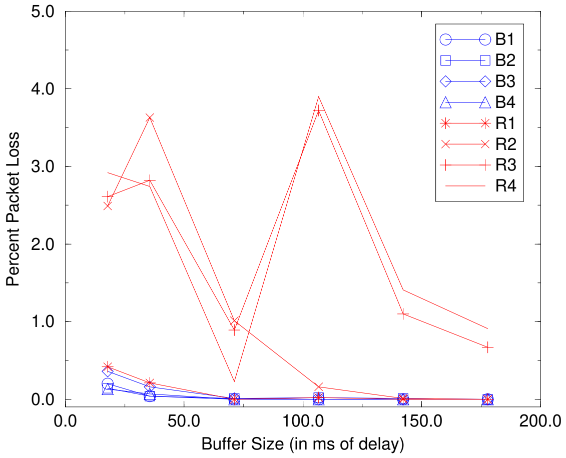
Figures 13(a) and (b) show the throughput and packet loss rates at the bottleneck link across a range of workloads. The throughput measures the rate at which packets are forwarded through the congested interface while the packet loss rate measures the ratio of the number of packets dropped at the queue and



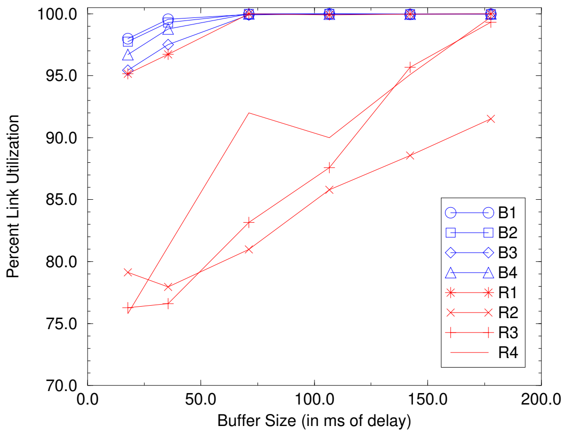
(a) Loss rates (1000 sources)



(b) Link utilization (1000 sources)



(c) Loss rates (4000 sources)



(d) Link utilization (4000 sources)

Figure 11: Performance of RED and BLUE with ECN timeouts

the total number of packets received at the queue. In each experiment, throughput and packet loss rates were measured over five 10-second intervals and then averaged. Note that the TCP sources used in the experiment do not implement ECN timeouts. As Figure 13(a) shows, both the BLUE queue and the optimally configured RED queue maintain relatively high levels of throughput across all loads. However, since RED periodically allows the link to become underutilized, its throughput remains slightly below that of BLUE. As Figure 13(b) shows, RED sustains increasingly high packet loss as the number of connections is increased. Since aggregate TCP traffic becomes more aggressive as the number of connections increases, it becomes difficult for RED to maintain low loss rates. Fluctuations in queue lengths occur so abruptly that the RED algorithm oscillates between periods of sustained marking and packet loss to periods of minimal marking and link underutilization. In contrast, BLUE maintains relatively small packet loss rates across all loads. At higher loads, when packet loss is observed, BLUE maintains a marking probability which is approximately 1, causing it to mark every packet it forwards.

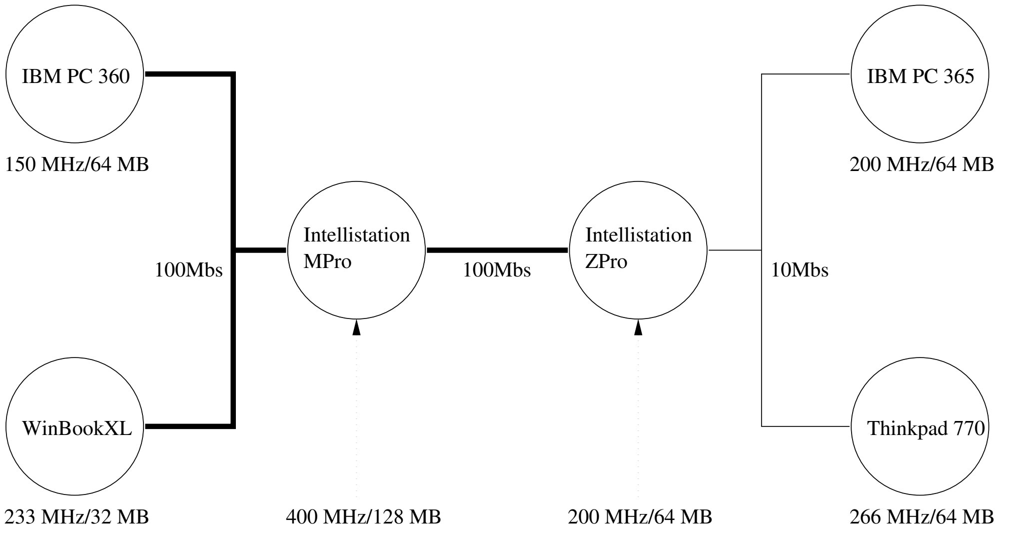
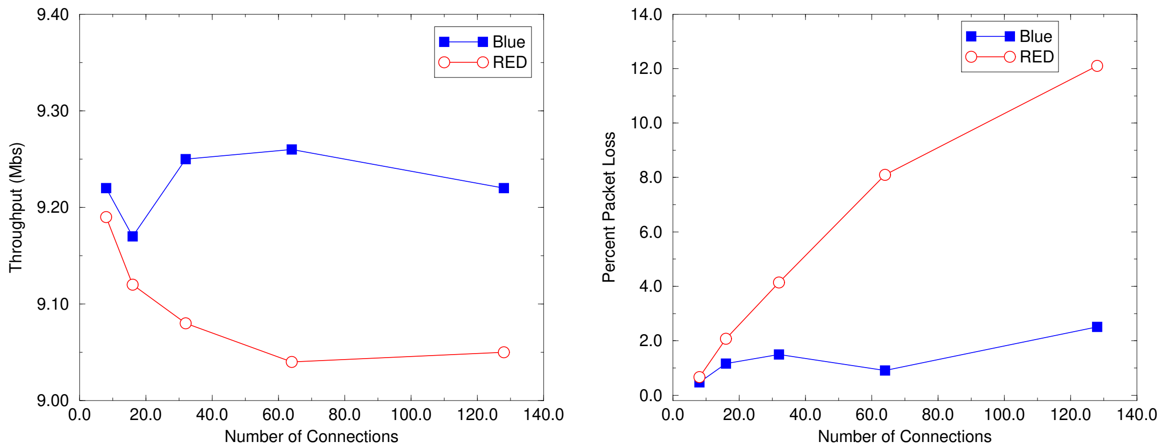


Figure 12: Experimental testbed



(a) Throughput

(b) Percent packet loss

Figure 13: Queue management performance

4 Stochastic Fair BLUE

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 [17, 23]. 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 [6, 18]. 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 section describes and evaluates *Stochastic Fair* BLUE (SFB), a novel technique for protecting TCP flows against non-responsive flows using the BLUE algorithm. SFB is highly scalable and enforces fairness using an extremely small amount of state and a small

```
B[l][n]: L x N array of bins (L levels, N bins per level)
enque()
        Calculate hash function values 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;
```

Figure 14: SFB algorithm

amount of buffer space.

4.1 The algorithm

Figure 14 shows the basic SFB algorithm. SFB is a FIFO queueing algorithm that identifies and rate-limits non-responsive flows 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. In addition, SFB 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 [20] (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 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

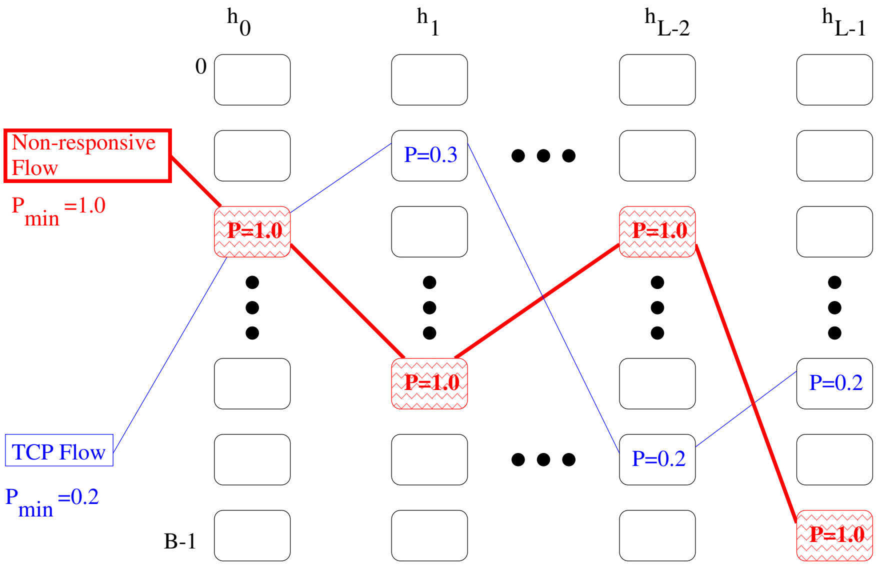


Figure 15: Example of SFB

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 [1]. 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 15 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. 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 the same techniques used for SFB to RED.

4.2 Evaluation

Using `ns`, the SFB algorithm was simulated in the same network as in Figure 4 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

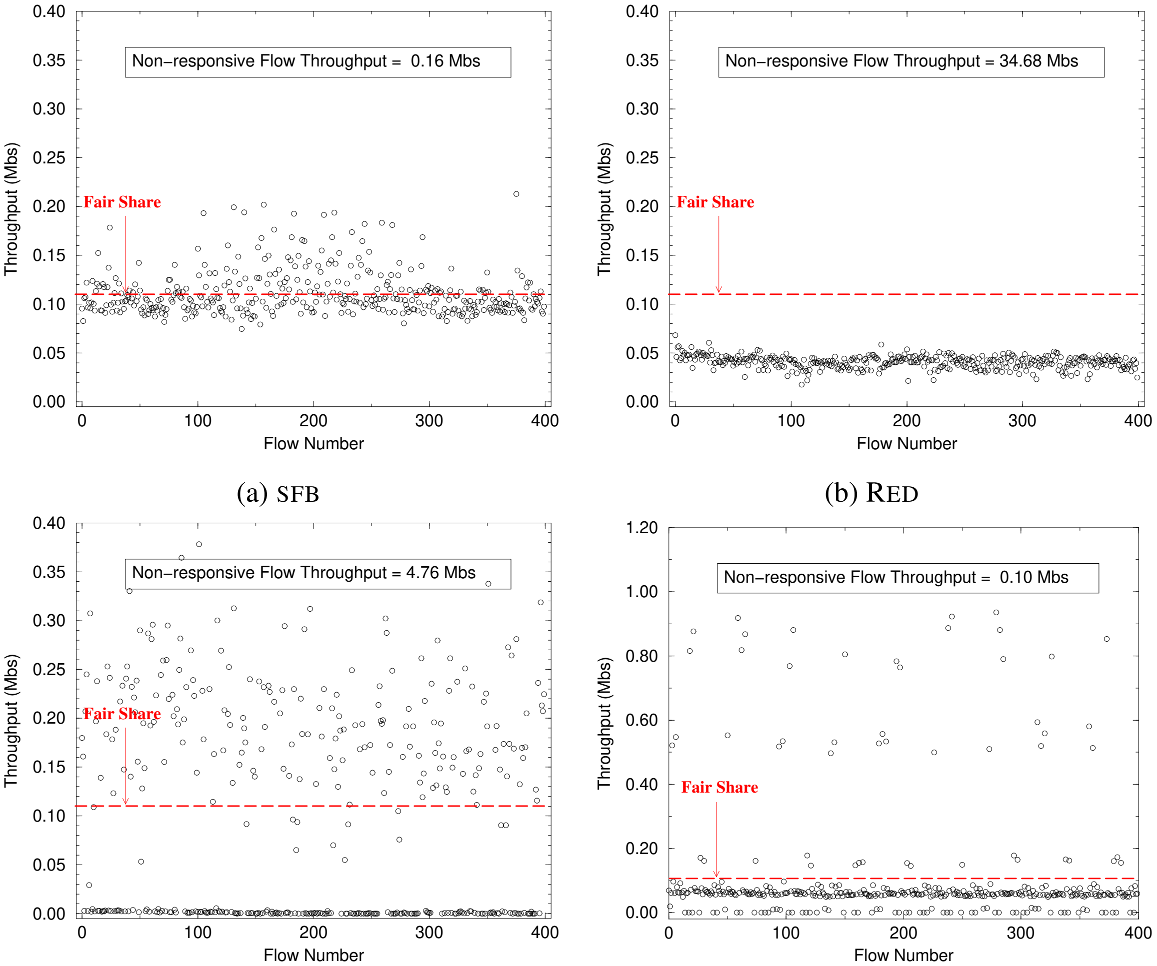
Packet Loss (Mbs)	$2Mbs$ non-responsive flow				$45Mbs$ non-responsive flow			
	SFB	RED	SFRED	SFQ+RED	SFB	RED	SFRED	SFQ+RED
Total	1.86	1.79	3.10	3.60	44.85	13.39	42.80	46.47
Non-responsive flow	1.85	0.03	0.63	1.03	44.84	10.32	40.24	43.94
All TCP flows	0.01	1.76	2.57	2.47	0.01	3.07	2.56	2.53

Table 3: SFB loss rates in Mbs (one non-responsive flow)

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 $0.16Mbs$.

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 3 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 3 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 16(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 16(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 16(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 4.1. Finally, Figure 16(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 induces a large number of retransmission timeouts across a subset of flows which causes significant amounts of unfairness [21]. Thus, through the course of the experiment, a few TCP flows are able to 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



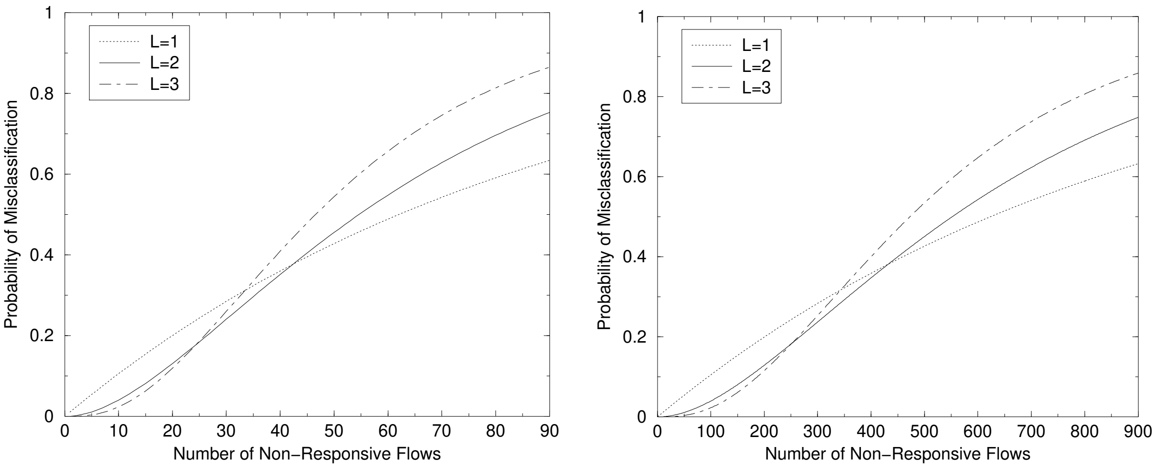
(a) SFB (b) RED (c) SFRED (d) SFQ+RED

Figure 16: Bandwidth of TCP flows ($45Mbs$ non-responsive flow)

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.

4.3 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



(a) 90 bins

(b) 900 bins

Figure 17: Probability of misclassification

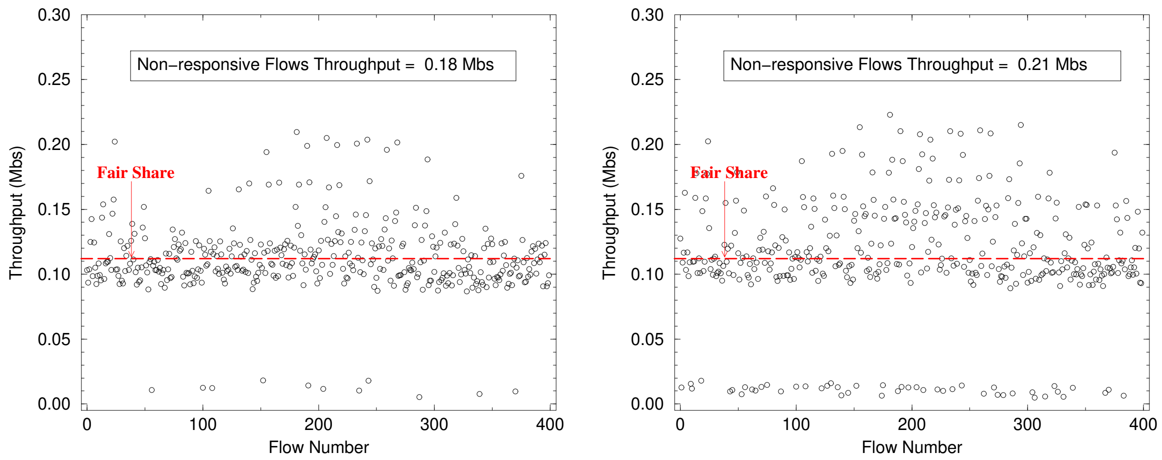
misclassified, increases. Clearly, misclassification limits the ability of SFB to protect well behaved TCP flows.

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.

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

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 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 17(a) shows the probability of misclassifying a flow when the total number of bins is fixed at 90. Figure 17(b) shows the same probability function when the total number of bins is fixed at 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.



(a) 4 non-responsive flows

(b) 8 non-responsive flows

Figure 18: Bandwidth of TCP flows using SFB

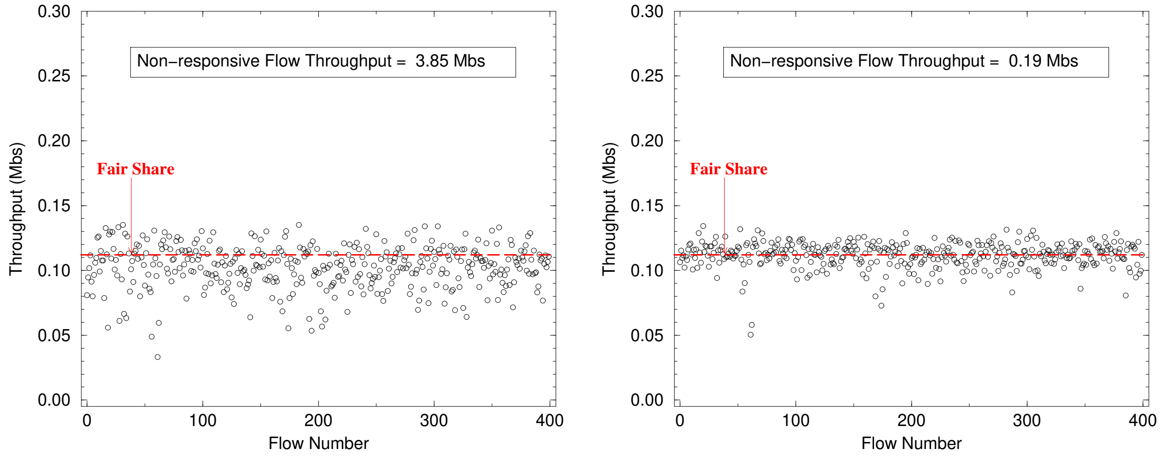
In order to evaluate the performance degradation of SFB as the number of non-responsive flows increases, Figure 18 shows the bandwidth plot of the 400 TCP flows when 4 and 8 non-responsive flows are present. In these experiments, each non-responsive flow transmits at a rate of $5Mbs$. As Equation (1) 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 flows (34) when 4 non-responsive flows are present and 16.12% of the TCP flows (64) when 8 non-responsive are present.

4.4 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 [16, 28] 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 exper-



(a) Moving hash

(b) Double buffered moving hash

Figure 19: Bandwidth of TCP flows using modified SFB algorithms

iment in Figure 18(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 19(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 18 where a significant number of TCP flows receive very little bandwidth.

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 Figure 19(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 hash functions are switched, the bins which have been warmed up are then used. Consequently, non-responsive flows are rate-limited right from the beginning. Figure 19(b) shows the performance of this approach. As the figure shows, the double buffered moving hash 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 4 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

	Loss Rates (in Mbs)			
	$10s$-$30s$	$30s$-$50s$	$50s$-$70s$	$70s$-$100s$
TCP Flows	0.402	0.358	0.260	0.324
Non-responsive Flow	4.866	4.849	4.898	4.863
Oscillating Flow	4.871	0.025	4.845	0.017
Total	10.139	5.232	10.003	5.204

Table 4: SFB loss rates (One non-responsive, One oscillating flow)

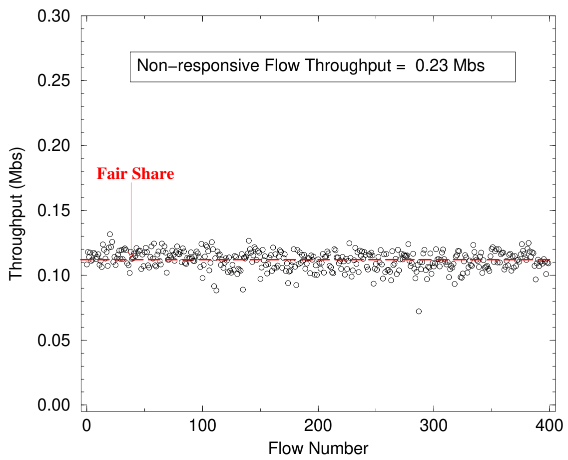
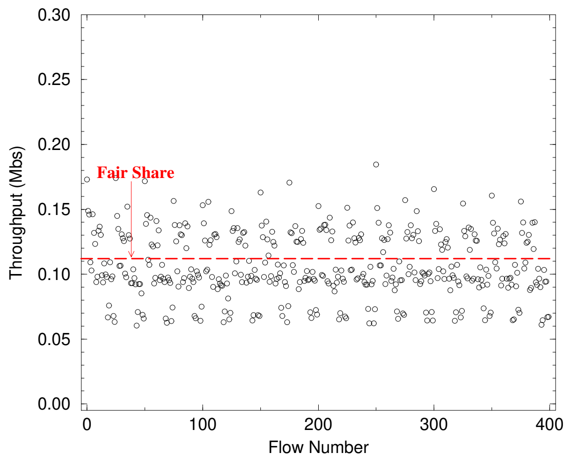


Figure 20: Bandwidth of TCP flows (One non-responsive, one oscillating flow)

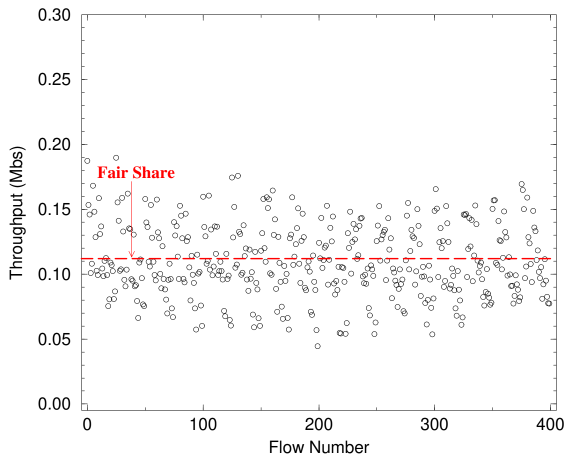
its transmission rate. The table also shows that when the second flow transmits at $5Mbs$, it observes a sustained packet loss rate as a large fraction of its packets are dropped by the queue. When the second flow cuts its transmission rate to a fair share of the link's bandwidth, it is reclassified 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 20 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.

4.5 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 probability 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.



(a) RED



(b) SFB

Figure 21: Bandwidth of TCP flows over varying round-trip times.

Figure 21 shows the performance of RED and SFB using the original network shown in Figure 4. 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.

5 Comparisons 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.

5.1 RED with Penalty Box

The RED with penalty box approach takes advantage of the fact that high bandwidth flows see proportionally larger amounts of packet loss. By keeping a finite log of recent packet loss events, this algorithm identifies flows which are non-responsive based on the log [19]. Flows which are identified as being non-responsive are then rate-limited using a mechanism such as class-based queueing [14]. 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 potentially 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 thus 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.

5.2 FRED

Another proposal for using RED mechanisms to provide fairness is Flow-RED (FRED) [18]. 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 exploits this property of FRED in order to circumvent its protection mechanisms. Note that SFB 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.

5.3 RED with per-flow Queueing

A RED-based, per-active flow approach has been proposed for providing fairness between flows [27]. 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 viable alternative for providing fairness efficiently.

5.4 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 4.2, 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.

5.5 Core-Stateless Fair Queueing

Core-Stateless Fair Queueing [26] (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 mechanism 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.

6 Conclusion and Future Work

This paper has demonstrated the inherent weakness of current active queue management algorithms which use queue occupancy in their algorithms. In order to address this problem, a fundamentally different queue management algorithm called BLUE has been designed and evaluated. BLUE uses the packet loss and link utilization history of the congested queue, instead of queue lengths to manage congestion. In addition to BLUE, this paper has proposed and evaluated SFB, a novel algorithm for scalably and accurately enforcing fairness amongst flows in a large aggregate. Using SFB, non-responsive flows can be identified and rate-limited using a very small amount of state.

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 4. Finally, the development of an "enhanced" BLUE queue management algorithm which is similar to "enhanced" RED [8, 9] 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] R. 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. Recommenda-

tions on Queue Management and Congestion Avoidance in the Internet. *RFC 2309*, April 1998.

[3] K. Cho. A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX Based Routers. *USENIX 1998 Annual Technical Conference*, June 1998.

[4] I. Cidon, R. Guerin, and A. Khamisy. Protective Buffer Management Policies. *IEEE/ACM Transactions on Networking*, 2(3), June 1994.

[5] S. Doran. RED Experience and Differentiated Queueing. In *NANOG Meeting*, June 1998.

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

[7] W. Feng, D. Kandlur, D. Saha, and K. Shin. Techniques for Eliminating Packet Loss in Congested TCP/IP Networks. In *UM CSE-TR-349-97*, October 1997.

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

[9] 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.

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

[11] S. Floyd. TCP and Explicit Congestion Notification. *Computer Communication Review*, 24(5):10–23, October 1994.

[12] S. Floyd and V. Jacobson. On Traffic Phase Effects in Packet-Switched Gateways. *Internetworking: Research and Experience*, 3(3):115–156, September 1992.

[13] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *ACM/IEEE Transactions on Networking*, 1(4):397–413, August 1993.

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

[15] R. Guerin, S. Kamat, V. Peris, and R. Rajan. Scalable QoS Provision Through Buffer Management. In *Proceedings of ACM SIGCOMM*, September 1998.

[16] IEEE 802.11 Working Group. IEEE 802.11 Standard, June 1997.

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

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

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

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

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

[22] Netperf. The Public Netperf Homepage: http://www.netperf.org/. The Public Netperf Homepage, 1998.

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

[24] K. K. Ramakrishan and R. Jain. A Binary Feedback Scheme for Congestion Avoidance in Computer Networks. *ACM Transaction on Computer Systems*, 8(2):158–181, May 1990. **Review:** *Computing Reviews*, December 1990.

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

[26] 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.

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

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

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