The BLUE Active Queue Management Algorithms

Wu-chang Feng, Kang G. Shin, *Fellow, IEEE*, Dilip D. Kandlur, *Member, IEEE*, and Debanjan Saha, *Member, IEEE*

***Abstract*—In order to stem the increasing packet loss rates caused by an exponential increase in network traffic, the IETF has been considering the deployment of active queue management techniques such as RED [14]. While active queue management can potentially reduce packet loss rates in the Internet, we show that current techniques are ineffective in preventing high loss rates. The inherent problem with these queue management algorithms is that they 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, implemented, and evaluated. BLUE uses packet loss and link idle events to manage congestion. Using both 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 based on Bloom filters [2] is described for enforcing fairness among a large number of flows. In particular, we propose and evaluate Stochastic Fair BLUE (SFB), a queue management algorithm which can identify and rate-limit nonresponsive flows using a very small amount of state information.**

***Index Terms*—Congestion control, fair queue, networks, queue management.**

I. 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 [19]. 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 [28]. 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) [12], [30], [31] along with active queue management techniques such as RED (Random Early Detection) [3], [14]. While ECN is necessary for eliminating packet loss in the Internet [9], we show 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 ($\min_{\rm th}$), packets are randomly dropped or marked with an 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 other active queue management techniques is that they rely on queue length as an estimator of congestion.[1] 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. From well-known results in queuing theory, it is only when packet interarrivals have a Poisson distribution that queue lengths directly relate to the number of active sources and thus the true level of congestion. Unfortunately, packet interarrival times across network links are decidedly non-Poisson. Packet interarrivals from individual sources are driven by TCP dynamics and source interarrivals themselves are heavy-tailed in nature [21], [29]. This makes placing queue length at the heart of an active queue management scheme dubious. 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 [6], [34].

In light of the above observation, we propose a fundamentally different active queue management algorithm, called BLUE, 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

Manuscript received March 13, 2001; revised September 24, 2001; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor R. Cruz. The work of W. Feng and K. G. Shin was supported in part by an IBM Graduate Fellowship and by the Office of Naval Research under Grant N00014-99-1-0465.

W. Feng is with the Department of Computer Science and Engineering, Oregon Graduate Institute, Oregon Health and Science University, Beaverton, OR 97006 USA (e-mail: wuchang@cse.ogi.edu).

K. G. Shin is with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109-2122 USA (e-mail: kgshin@umich.edu).

D. D. Kandlur is with IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA (e-mail: kandlur@us.ibm.com).

D. Saha is with Tellium, Inc., West Long Branch, NJ 07764 USA (e-mail: dsaha@tellium.com).

Publisher Item Identifier 10.1109/TNET.2002.801399.

[1]We note that at the time of initial publication [10], BLUE was the only active queue management which did not use queue length. Subsequent algorithms [1], [17], [20] have also shown the benefits of decoupling queue length from congestion management.

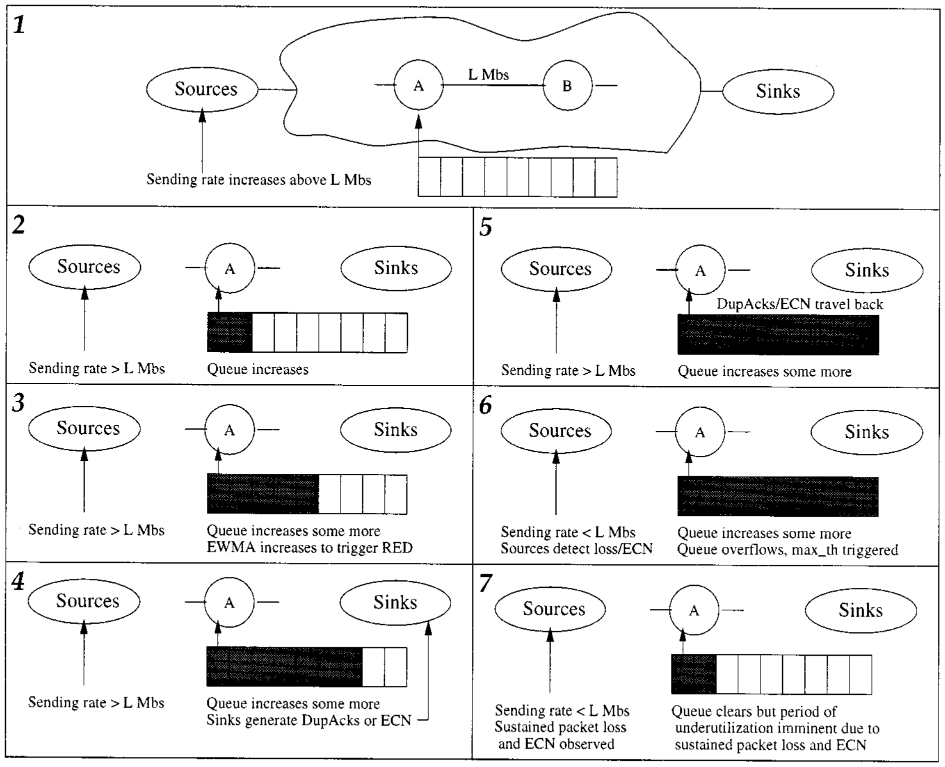

1
Sources
A
L Mbs
B
Sinks
Sending rate increases above L Mbs
2
Sending rate > L Mbs
Queue increases
3
Sending rate > L Mbs
Queue increases some more
EWMA increases to trigger RED
4
Sending rate > L Mbs
Queue increases some more
Sinks generate DupAcks or ECN
5
DupAcks/ECN travel back
Sending rate > L Mbs
Queue increases some more
6
Sending rate < L Mbs
Sources detect loss/ECN
Queue increases some more
Queue overflows, max_th triggered
7
Sending rate < L Mbs
Sustained packet loss
and ECN observed
Queue clears but period of
underutilization imminent due to
sustained packet loss and ECN

Fig. 1. RED example.

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 both simulation and experimentation, we demonstrate 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 II gives a description of RED and shows why it is ineffective at managing congestion. Section III describes BLUE and provides a detailed analysis and evaluation of its performance based on simulation as well as controlled experiments. Section IV describes and evaluates Stochastic Fair BLUE (SFB), an algorithm based on BLUE which scalably enforces fairness amongst a large number of connections. Section V compares SFB to other approaches which have been proposed to enforce fairness amongst connections. Finally, Section VI concludes with a discussion of future work.

II. Background

One of the biggest problems with TCP's congestion control algorithm over drop-tail queues is that sources reduce their transmission rates only after detecting packet loss due to queue overflow. Since a 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 must also 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 [8], [9]. Bursty traffic often defeats the active queue management techniques used by RED since queue lengths grow and shrink rapidly, well before RED can react. Fig. 1 shows a simplified pictorial example of how RED functions under this congestion scenario.

The congestion scenario presented in Fig. 1 occurs when a large number of TCP sources are active 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 $\min_{\rm th}$ 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

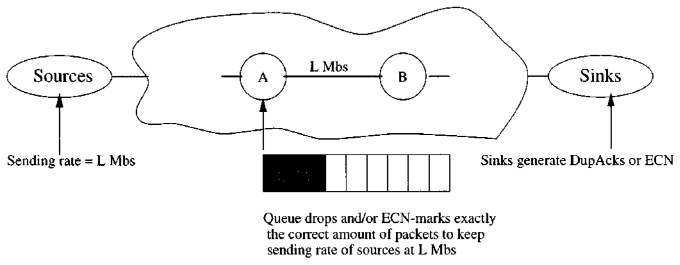

Sources
A
L Mbs
B
Sinks
Sending rate = L Mbs
Sinks generate DupAcks or ECN
Queue drops and/or ECN-marks exactly the correct amount of packets to keep sending rate of sources at L Mbs

Fig. 2. Ideal scenario.

```
Upon packet loss (or $Q_{len} > L$) event:
    if ( (now - $last\_update$) > $freeze\_time$ )
        $p_m := p_m + \delta_1$
        $last\_update$ := now
Upon link idle event:
    if ( (now - $last\_update$) > $freeze\_time$ )
        $p_m := p_m - \delta_2$
        $last\_update$ := now
```

Fig. 3. BLUE algorithm.

packets with their ECN bits set. In response, duplicate acknowledgment and/or TCP-based ECN signals are sent back to the sources. At $t = 5$, the duplicate acknowledgment 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 [8], [9] 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 [34]. 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 large amounts of buffer adds considerable end-to-end delay and delay jitter. This severely impairs the ability to run interactive applications. In addition, the abundance of deployed routers which have limited memory resources makes this solution undesirable.

Fig. 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.

III. BLUE

In order to remedy the shortcomings of RED, we propose, implement, and evaluate a fundamentally different queue management algorithm called BLUE. Using both simulation and experimentation, we show that BLUE overcomes 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 Fig. 2 in almost all scenarios, even when used with very small buffers.

A. 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 sharp 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. Fig. 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 queueing 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 paper fix *freeze_time* as a constant, this value should be randomized in order to avoid global synchronization [13]. The other parameters used (δ_1 and δ_2) 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, δ_1 is set significantly larger than δ_2. 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 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. It is relatively simple process to configure BLUE to meet the goals of controlling congestion. The first parameter, *freeze_time*, should be set based on the effective round-trip times of connections multiplexed across the link in order to allow any changes in the marking probability to reflect back on to the end sources

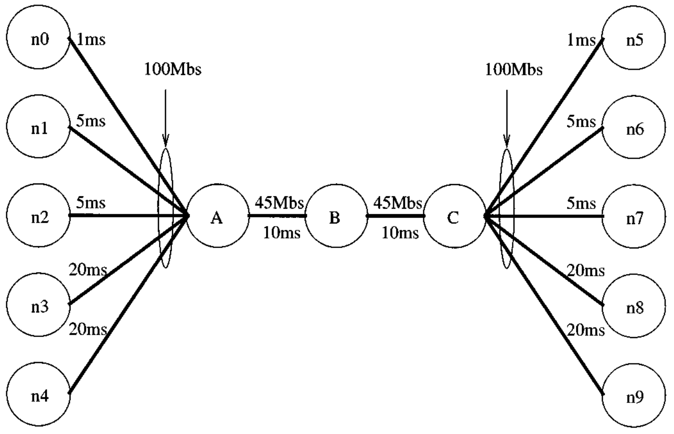


Fig. 4. Network topology.

before additional changes are made. For long-delay paths such as satellite links, *freeze_time* should be increased to match the longer round-trip times. The second set of parameters δ_1 and δ_2 are set to give the link the ability to effectively adapt to macroscopic changes in load across the link at the connection level. For links where extremely large changes in load occur only on the order of minutes, δ_1 and δ_2 should be set in conjunction with *freeze_time* to allow p_m to range from 0 to 1 on the order of minutes. This is in contrast to current queue length approaches where the marking and dropping probabilities range from 0 to 1 on the order of milliseconds even under constant load. Over typical links, using *freeze_time* values between 10 and 500 ms and setting δ_1 and δ_2 so that they allow p_m to range from 0 to 1 on the order of 5 to 30 s will allow the BLUE control algorithm to operate effectively. Note that, while BLUE algorithm itself is extremely simple, it provides a significant performance improvement even when compared to a RED queue which has been reasonably configured.

B. Packet Loss Rates Using RED and BLUE

In order to evaluate the performance of BLUE, a number of simulation experiments were run using ns [23] over a small network shown in Fig. 4. Using this network, Pareto on/off sources with mean on-times of 2 s and mean off-times of 3 s 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 1 s of simulation, and used 1 kB packets. Packet loss statistics were then measured after 100 s of simulation for 100 s. Loss statistics were also measured for RED using the same network and under identical conditions. For the RED queue, $\min_{\rm th}$ and $\max_{\rm th}$ 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 [9]. 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 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

TABLE I
RED CONFIGURATIONS

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

TABLE II
BLUE CONFIGURATIONS

Configuration	$freeze\_time$	δ_1	δ_2
$B1$	$10ms$	0.0025	0.00025
$B2$	$100ms$	0.0025	0.00025
$B3$	$10ms$	0.02	0.002
$B4$	$100ms$	0.02	0.002

algorithm becomes decoupled from the queue length and thus acts more like BLUE. Table I shows the configurations used for RED; Table II shows the configurations used for BLUE. For the BLUE experiments, δ_1 and δ_2 are set so that δ_1 is an order of magnitude larger than δ_2. Using these values, the *freeze_time* is then varied between 10 and 100 ms. Additional simulations using a wider range of values were also performed and showed similar results.

Fig. 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 100 to 1000 kB. This corresponds to queueing delays which range from 17.8 and 178 ms as shown in the figure. In all experiments, the link remains over 99.9% utilized. As Fig. 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 [34]. 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 Fig. 5(a) is that it shows a significant dip in loss rates at a buffering delay of around 80 ms. This occurs because of a special operating point of RED when the average queue length stays above $\max_{\rm th}$ 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 $\max_{\rm th}$. 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 $\max_{\rm th}$. 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

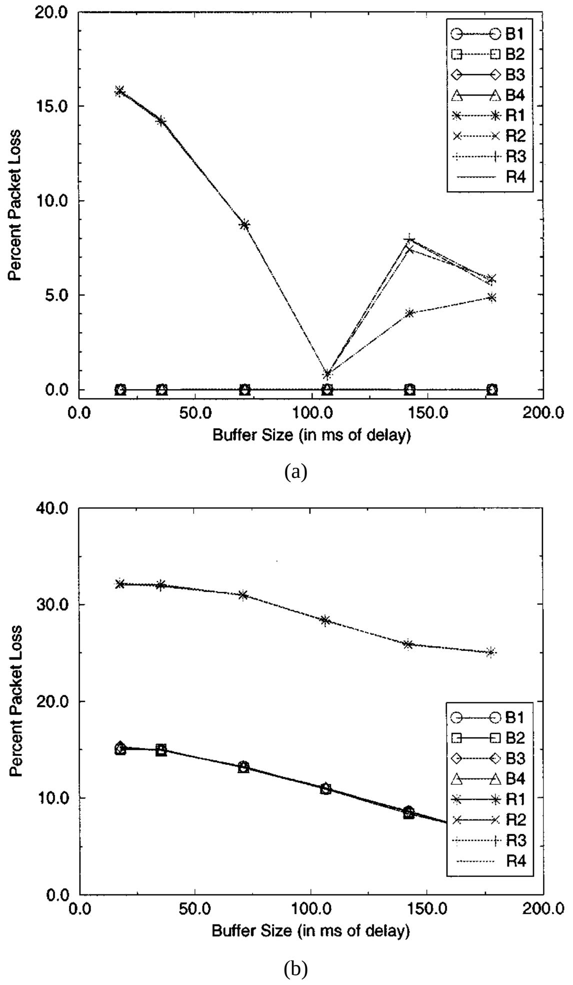


Fig. 5. Packet loss rates of RED and BLUE. (a) 1000 sources. (b) 4000 sources.

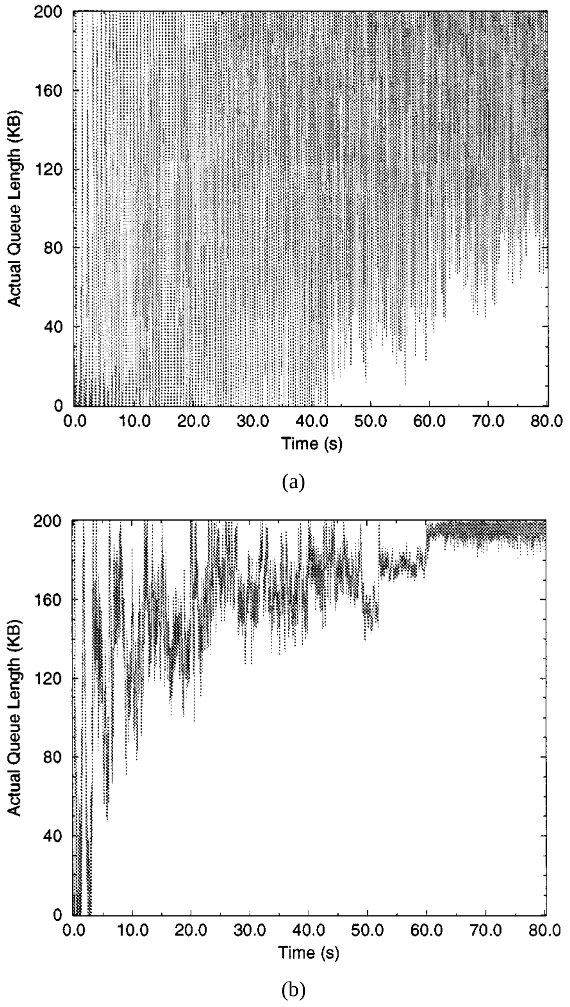


Fig. 6. Queue length plots of (a) RED and (b) BLUE.

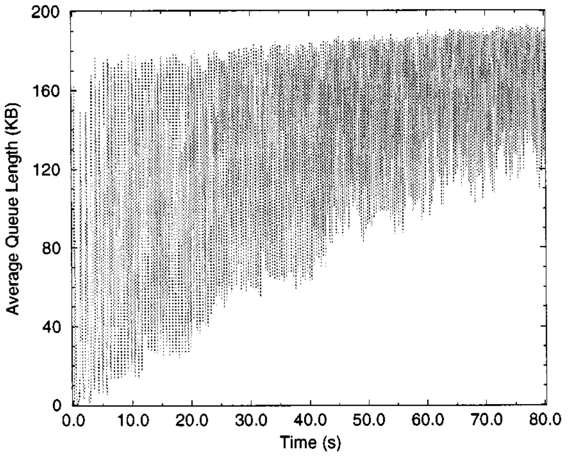
with deterministic marking. When combined, this leads again to fluctuations in queue length. At a load which is perfectly selected, the average queue length of RED can remain at $\max_{th}$ and the queue can avoid packet loss and prevent queue fluctuations by marking every packet. As Fig. 5(b) shows, 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 17.8 ms 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 III-D shows how this can significantly influence the performance of both RED and BLUE.

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 [5], [16] 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.

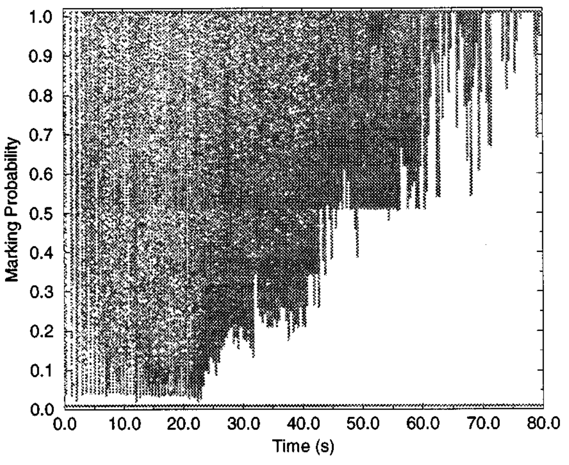
C. *Understanding* BLUE

To fully understand the difference between the RED and BLUE algorithms, Fig. 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 s. As Fig. 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 rates. In contrast, as Fig. 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.

Fig. 7 plots the average queue length ($Q_{\rm ave}$) and the marking probability $p_b/(1 - \text{count} \times p_b)$ of RED throughout the experiment. The average queue length of RED contributes directly



(a)



(b)

Fig. 7. Marking behavior of RED. (a) Q_{ave}. (b) $p_b/(1 - \text{count} \times p_b)$

to its marking probability since p_b is a linear function of Q_{ave} ($p_b = \max_p \times ((Q_{\text{ave}} - \min_{\text{th}})/(\max_{\text{th}} - \min_{\text{th}}))$). As shown in Fig. 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 Fig. 7(b), fluctuates considerably as well. In contrast, Fig. 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 loss is when every packet is being marked, but the offered load still overwhelms the bottleneck link. As described earlier, this occurs at $t = 60$ s 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.

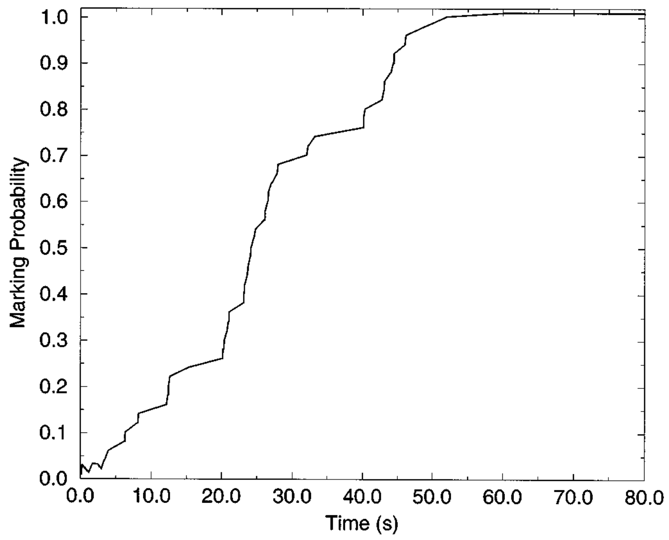


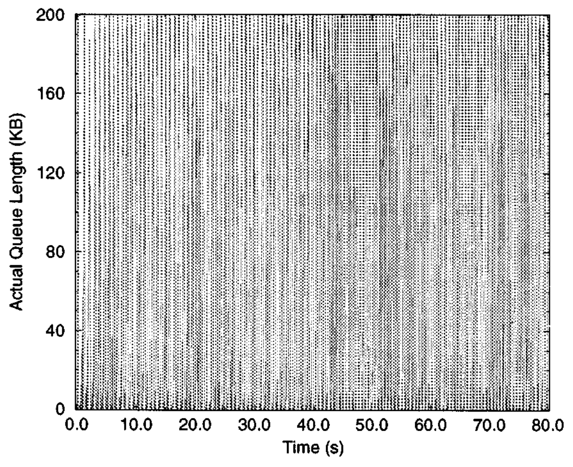
Fig. 8. Marking behavior of BLUE (p_m).

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 Fig. 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 Fig. 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.

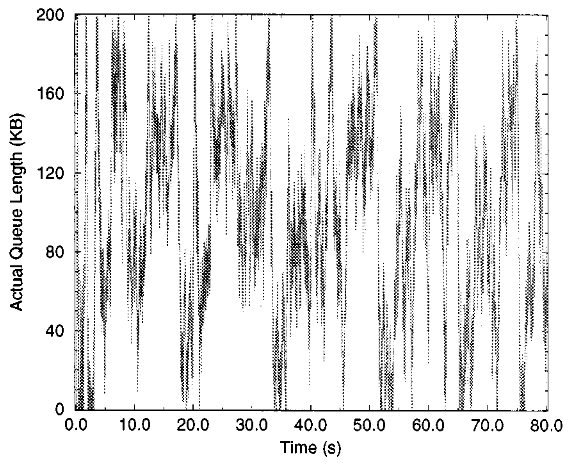
Another goal of RED is to eliminate biases against bursty sources in the network. This is done by limiting the queue occupancy so that there is always room left in the queue to buffer transient bursts. In addition, the marking function of RED takes into account the last packet marking time in its calculations 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 (Fig. 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.

D. 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. Fig. 9 shows

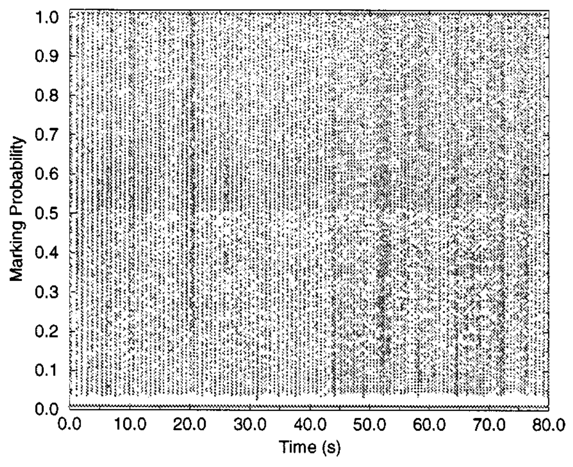


(a)

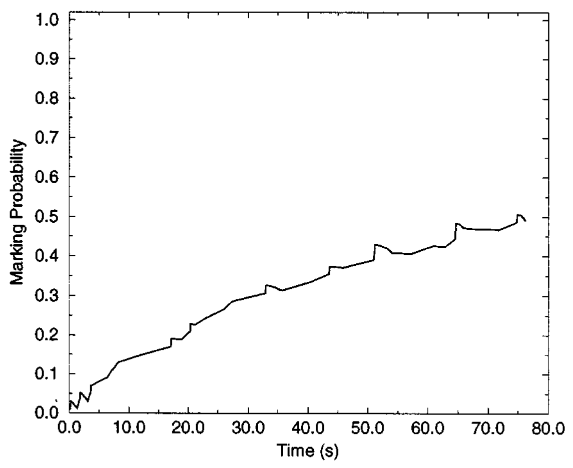


(b)

Fig. 9. Queue length plots of (a) RED and (b) BLUE with ECN timeouts.



(a)



(b)

Fig. 10. Marking behavior with ECN timeouts. (a) $p_b/(1 - \text{count} \times p_b)$ of RED. (b) p_m of BLUE.

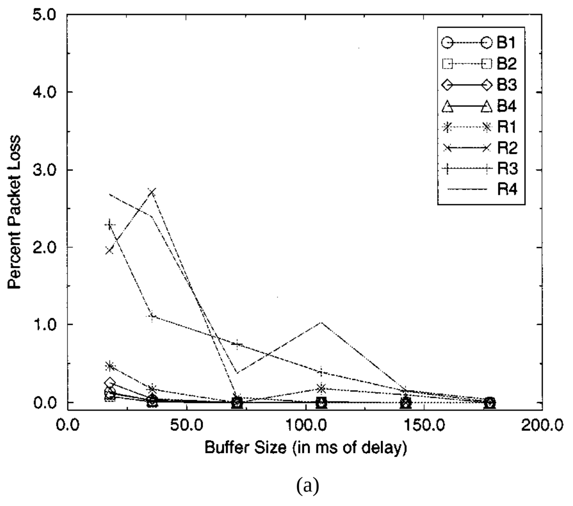
the queue length plot of RED and BLUE using the same experiments as in Section III-B, but with TCP sources enabled with ECN timeouts. Fig. 9(a) shows that, by deterministically marking packets at $\max_{\text{th}}$, RED oscillates between periods of packet loss and periods of underutilization as described in Section II. Note that this is in contrast to Fig. 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 $\max_{\text{th}}$ 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. Fig. 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.

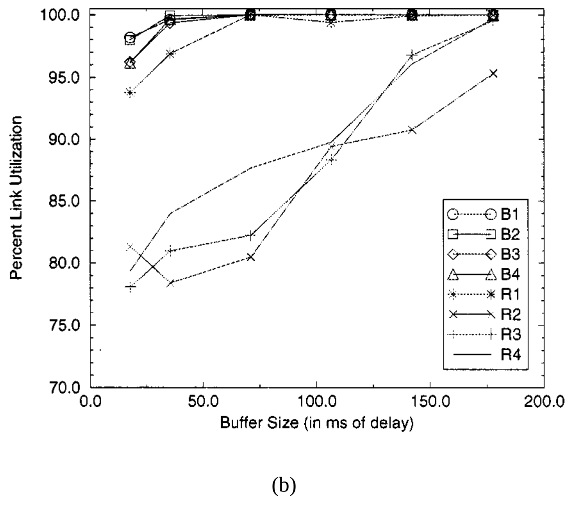
Fig. 10 shows the corresponding marking behavior of both RED and BLUE in the experiment. As the figure shows, BLUE maintains a steady marking rate which changes as the workload is changed. On the other hand, RED's calculated marking probability fluctuates from 0 to 1 throughout the experiment. When the queue is fully occupied, RED overmarks and drops packets causing a subsequent period of underutilization as described in Section II. 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.

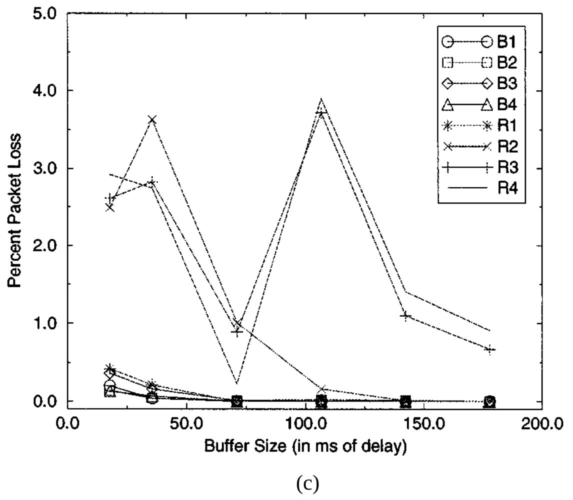
Fig. 11 shows how ECN timeouts impact the performance of RED and BLUE. The figure shows the loss rates and link utilization using the 1000- and 4000-connection experiments in Section III-B. As the figure shows, BLUE maintains low packet loss rates and high link utilization across all experiments. The figure also shows that the use of ECN timeouts allows RED to reduce the amount of packet loss in comparison to Fig. 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 management behave more like BLUE.

E. Implementation

In order to evaluate BLUE in a more realistic setting, it has been implemented in FreeBSD 2.2.7 using ALTQ [4]. In this







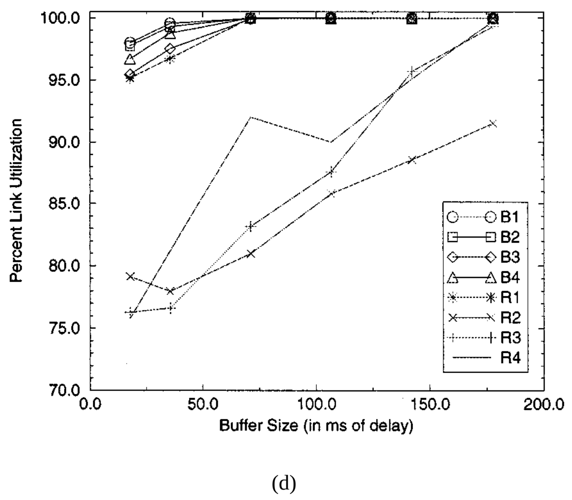


Fig. 11. Performance of RED and BLUE with ECN timeouts. (a) Loss rates (1000 sources). (b) Link utilization (1000 sources). (c) Loss rates (4000 sources). (d) Link utilization (4000 sources).

implementation, ECN uses two bits of the type-of-service (ToS) field in the IP header [31]. 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 Fig. 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` [26] 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 50 kB which corresponds to a queueing delay of about 40 ms. For the experiments with RED, a configuration with a $\min_{th}$ of 10 kB, a $\max_{th}$ of 40 kB, a $\max_p$ of 1, and a w_q of 0.002 was used. For the experiments with BLUE, a δ_1 of 0.01, a δ_2 of 0.001, and a *freeze_time* of 50 ms 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

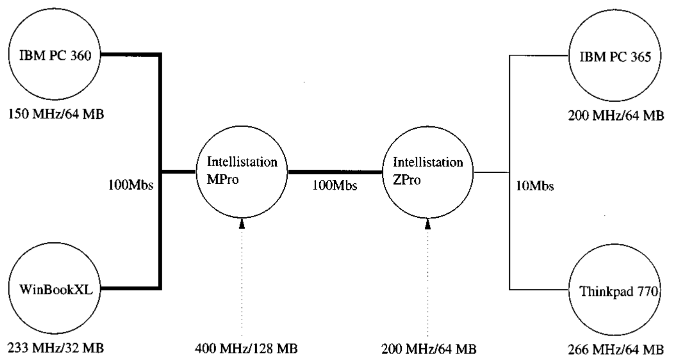


Fig. 12. Experimental testbed.

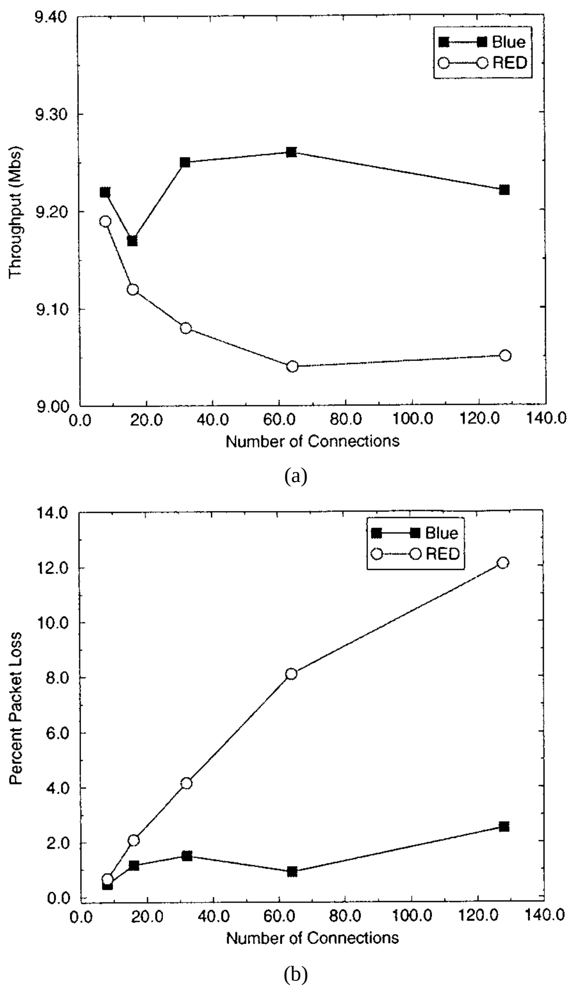


Fig. 13. Queue management performance. (a) Throughput. (b) Percent packet loss.

using the BLUE modifications on the intermediate routers. In all of the experiments, the sustained throughput was always above 80 Mb/s.

Fig. 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 the total number of packets received at the queue. In each experiment, throughput and packet loss rates were measured over five 10-s intervals and then averaged. Note that the TCP sources used in the experiment do not implement ECN timeouts. As Fig. 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 Fig. 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.

IV. 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 [19], [28]. In order to address the problem of nonresponsive flows, a lot of work has been done to provide routers with mechanisms for protecting against them [7], [22], [27]. The idea behind these approaches is to detect nonresponsive 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 based on Bloom filters [2] for protecting TCP flows against nonresponsive flows. Based on the BLUE algorithm. SFB is highly scalable and enforces fairness using an extremely small amount of state and a small amount of buffer space.

A. The Algorithm

Fig. 14 shows the basic SFB algorithm. SFB is a FIFO queueing algorithm that identifies and rate-limits nonresponsive 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, via its connection ID *(Source address, Destination address, Source port, Destination port, Protocol)*, 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 [24] (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 nonresponsive 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 nonresponsive flows, however, unless the number of nonresponsive 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 nonresponsive 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 nonresponsive flow and is then rate-limited. Note that this approach is akin

```
$init()$
    $B[l][n]$: Allocate $L \times N$ array of bins
    ($L$ levels, $N$ bins per level)

$enque()$
    Calculate hashes $h_0, h_1, \ldots, h_{L-1}$;
    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. 14. SFB algorithm.

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 nonresponsive flows which are multiplexed across the link [2]. When a nonresponsive 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 nonresponsive 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. Fig. 15 shows an example of how SFB works. As the figure shows, a nonresponsive 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 nonresponsive 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 nonresponsive. On the other hand, since the minimum marking probability of the nonresponsive flow is 1.0, it is identified as being nonresponsive and rate-limited.

Note that just as BLUE's marking probability can be used in SFB to provide protection against nonresponsive flows, it is also possible to apply Adaptive RED's $\max_p$ parameter [9] 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 nonresponsive 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.

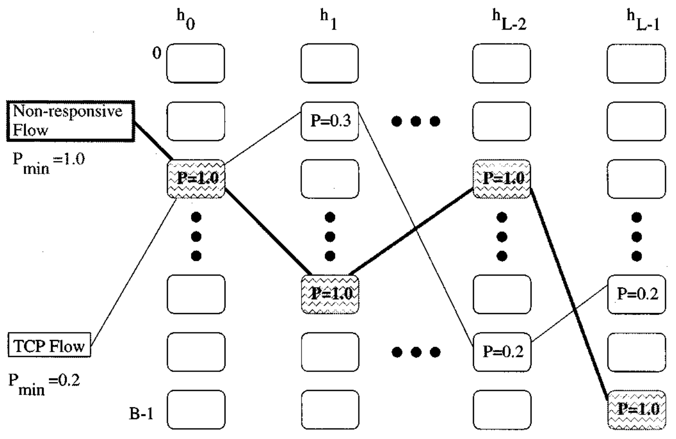


Fig. 15. Example of SFB.

B. Evaluation

Using ns, the SFB algorithm was simulated in the same network as in Fig. 4 with the transmission delay of all of the links set to 10 ms. The SFB queue is configured with 200 kB 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 1/23 of the available buffer space. Note that, by allocating more than 1/23 of 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 nonresponsive flows to 0.16 Mb/s.

In the experiments, 400 TCP sources and one nonresponsive, constant rate source are run for 100 s from randomly selected nodes in (n_0, n_1, n_2, n_3, and n_4) to randomly selected nodes in (n_5, n_6, n_7, n_8, and n_9). In one experiment, the nonresponsive flow transmits at a rate of 2 Mb/s while in the other, it transmits at a rate of 45 Mb/s. Table III shows the packet loss observed in both experiments for SFB. As the table shows, for both experiments, SFB performs extremely well. The nonresponsive 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 III also shows the performance of RED. In contrast to SFB, RED allows the nonresponsive 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 nonresponsive flow, however, it cannot fully protect the TCP sources from packet loss since it has a difficult time discerning nonresponsive 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 (414 kB), making each queue equally sized at 9 kB. For each bin in the SFQ, the RED algorithm was applied with $\min_{th}$ and $\max_{th}$ values set at 2 and 8 kB, respectively. As the table shows, SFQ with RED does an adequate job of protecting TCP flows from the nonresponsive flow. However,

TABLE III
SFB LOSS RATES IN Mb/s (ONE NONRESPONSIVE FLOW)

	2Mbps non-responsive flow				45Mbps non-responsive flow			
Packet Loss (Mbps)	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

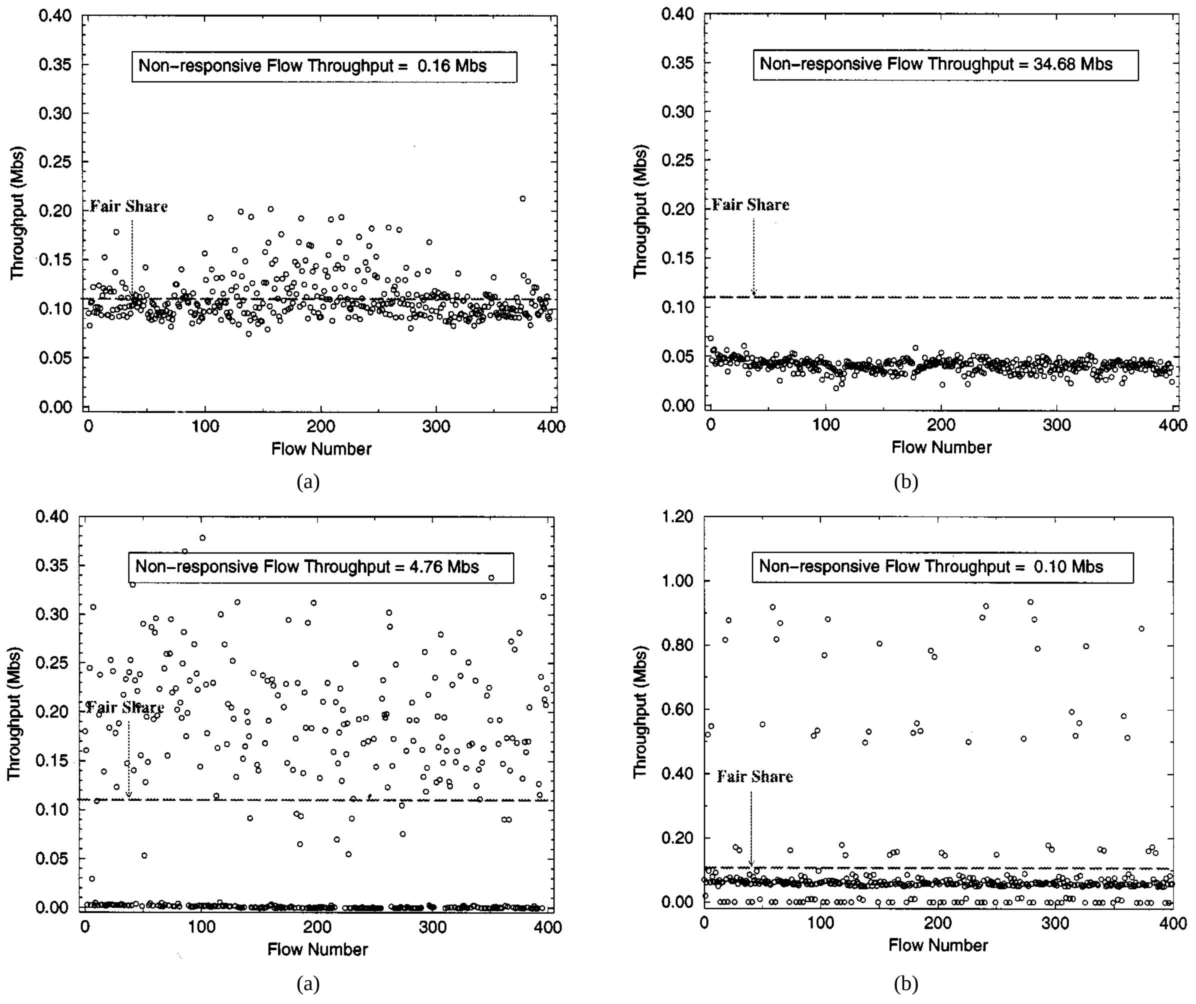


Fig. 16. Bandwidth of TCP flows (45 Mb/s nonresponsive flow). (a) SFB. (b) RED. (c) SFRED. (d) SFQ + RED.

in this case, partitioning the buffers into such small sizes causes a significant amount of packet loss to occur due to RED's inability to operate properly with small buffers. 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 nonresponsive flow has on TCP performance, Fig. 16(a) plots the throughput of all 400 TCP flows using SFB when the nonresponsive flow sends at a 45 Mb/s 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 nonresponsive flow is rate-limited to well below its transmission rate. In contrast, Fig. 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 nonresponsive flow is allowed to grab a large fraction of the bottleneck link bandwidth. Fig. 16(c) shows that while SFRED does succeed in rate-limiting the nonresponsive 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 IV-A. Finally, Fig. 16(d) shows that while SFQ with RED can effectively rate-limit the nonresponsive 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 [25]. Thus, through the course of the experiment, a few TCP flows are able to grab a disproportionate amount of the bandwidth

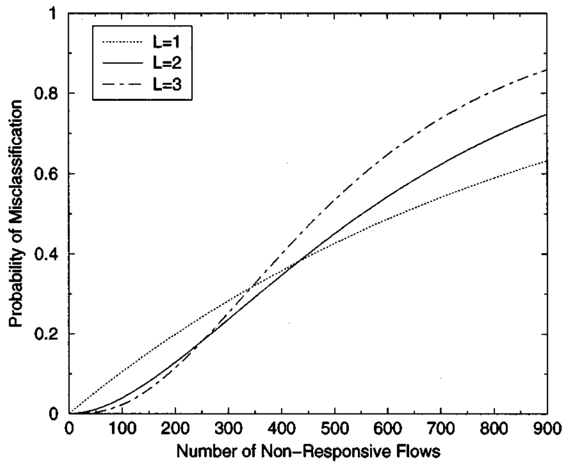


Fig. 17. Probability of misclassification using 900 bins.

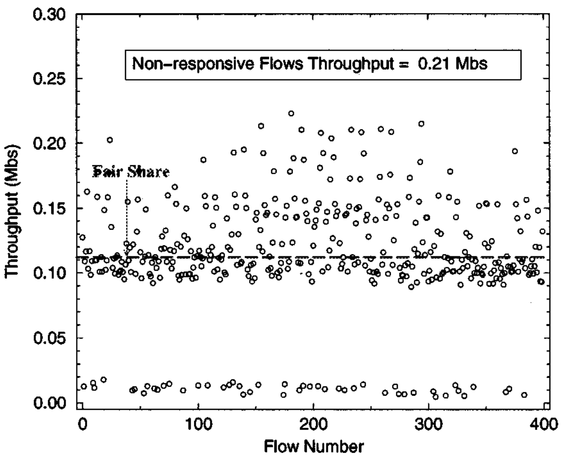


Fig. 18. Bandwidth of TCP flows using SFB with eight nonresponsive flows.

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 1/46 of the 400 flows to be mapped into the same queue as the nonresponsive 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.

C. Limitations of SFB

While it is clear that the basic SFB algorithm can protect TCP-friendly flows from nonresponsive 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 nonresponsive 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 nonresponsive flow without impacting the performance of any of the individual TCP flows. As the number of nonresponsive 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, (1) gives a closed-form expression of the probability that a well-behaved TCP flow gets misclassified as being nonresponsive as a function of number of levels (L), the number of bins per level (B), and the number of nonresponsive/malicious flows (M), respectively

$$p = \left[1 - \left(1 - \frac{1}{B}\right)^M\right]^L. \qquad (1)$$

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 (1), it is possible to optimize the performance of SFB given *a priori* information about its operating environment. Suppose the number of simultaneously active nonresponsive 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 (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. Fig. 17 shows the probability of misclassifying a flow when the total number of bins is fixed at 900. In this figure, the number of levels used in SFB along with the number of nonresponsive flows are varied. As the figures show, when the number of nonresponsive 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 nonresponsive 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 nonresponsive flow pollutes a larger number of bins. For example, using a single level SFB queue with 90 bins, a single nonresponsive flow pollutes only one bin. Using a two-level SFB queue with each level containing 45 bins, the number of effective bins is 45 $\times$ 45 (2025). However, a single nonresponsive flow pollutes two bins (one per level). Thus, the advantage gained by the two-level SFB queue is lost when additional nonresponsive flows are added, as a larger fraction of bins become polluted compared to the single-level situation.

In order to evaluate the performance degradation of SFB as the number of nonresponsive flows increases, Fig. 18 shows the bandwidth plot of the 400 TCP flows when eight nonresponsive flows are present. In these experiments, each nonresponsive flow transmits at a rate of 5 Mb/s. As (1) predicts, in an SFB configuration that contains two levels of 23 bins, 8.96% (36) of the TCP flows are misclassified when eight nonresponsive flows are present. When the number of nonresponsive flows approaches N, the performance of SFB deteriorates quickly as an increasing number of bins at each level becomes polluted. In the case of eight nonresponsive flows, approximately six 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 nonresponsive 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 16.12% of the TCP flows (64) when eight nonresponsive flows are present.

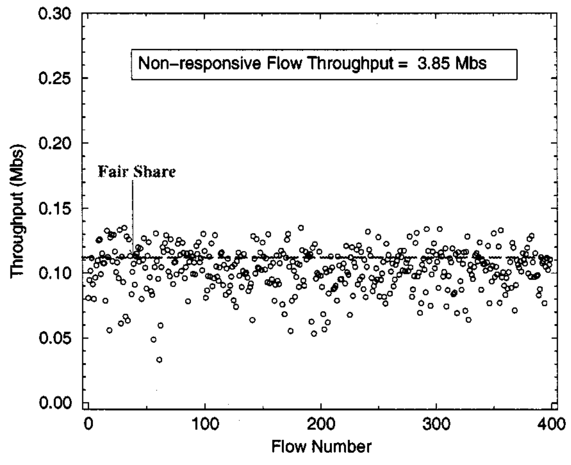
D. SFB With Moving Hash Functions

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 nonresponsive flows become responsive and to reclassify them when they do.

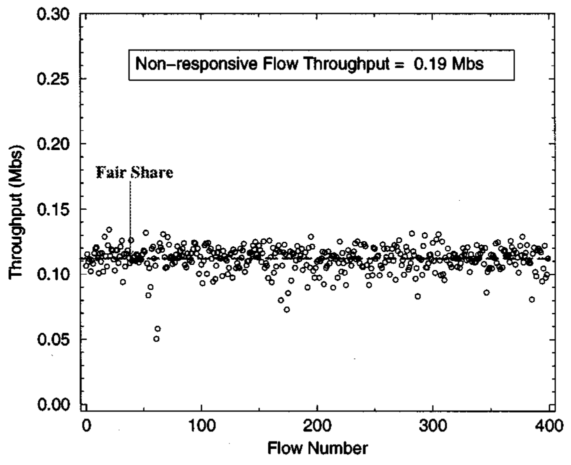
The idea behind SFB with moving hash functions is to periodically or randomly reset the bins and change the hash functions. A nonresponsive flow will continually be identified and rate-limited regardless of the hash function used. However, by changing the hash function, responsive TCP flows that happen 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 [18], [33] 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 Fig. 19(b). In this experiment, 8 nonresponsive flows along with 400 responsive flows share the bottleneck link. To protect against continual misclassification, the hash function is changed every 2 s. Fig. 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 Fig. 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, nonresponsive flows are temporarily placed on "parole." That is, nonresponsive 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. 19(a) shows, nonresponsive flows are allowed to consume 3.85 Mb/s 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 nonresponsive, 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, nonresponsive flows are rate-limited right from the beginning. Fig. 19(b) shows the performance of this approach. As the figure shows, the double buffered moving hash effectively controls the bandwidth of the nonresponsive flows and affords the TCP flows a very high level of protection. Note that one of the advantages of the moving hash function is that it can quickly react to nonresponsive 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. Additional experiments show that this algorithm allows for quick adaptation to flow behavior [11].



(a)



(b)

Fig. 19. Bandwidth of TCP flows using modified SFB algorithms.(a) Moving hash. (b) Double buffered moving hash.

V. Comparisons to Other Approaches

A. RED With a 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 nonresponsive based on the log [7]. Flows which are identified as being nonresponsive are then rate-limited using a mechanism such as class-based queueing [15]. While this approach may be viable under certain circumstances, it is unclear how the algorithm performs in the face of a large number of nonresponsive flows. Unless the packet loss log is large, a single set of high bandwidth flows can potentially dominate the loss log and allow other, nonresponsive flows to go through without rate-limitation. In addition, flows which are classified as nonresponsive 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 nonresponsive. 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. Stabilized RED

Stabilized RED is a another approach to detecting nonresponsive flows [27]. In this case, the algorithm keeps a finite log of recent flows it has seen. The idea behind this is that nonresponsive flows will always appear in the log multiple times and can be singled out for punishment. Unfortunately, for a large number of flows, using the last M flows can fail to catch nonresponsive flows. For instance, consider a single nonresponsive flow sending at a constant rate of 0.5 Mb/s in an aggregate consisting of 1000 flows over a bottleneck link of 100 Mb/s where a fair share of bandwidth is 0.1 Mb/s. In order to ensure that the nonresponsive flow even shows up in the last M flows seen, M needs to be at least 200 or 20% of the total number of flows. In general, if there are a total of N flows and a nonresponsive flow is sending at X times the fair share, M needs to be at least N/X in order to catch the flow. The SFB algorithm, on the other hand, has the property that the state scales with the number of nonresponsive flows. To ensure detection of the nonresponsive flow in the above situation, a static 10×3 SFB queue which keeps state on 30 bins or 3% of the total number of flows is sufficient. With the addition of mutating hash functions, an even smaller SFB queue can be used.

C. FRED

Another proposal for using RED mechanisms to provide fairness is Flow-RED (FRED) [22]. 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 buffer space to work well. Without sufficient buffer space, it becomes difficult for FRED to detect nonresponsive flows, as they may not have enough packets continually queued to trigger the detection mechanism. In addition, nonresponsive 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 behaviors. Because of this, SFB is better suited for protecting TCP flows against nonresponsive 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 nonresponsive 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 nonresponsive flows, FRED cannot make the distinction between N TCP flows multiplexed across a link versus N nonresponsive flows multiplexed across a link.

D. RED With Per-Flow Queueing

A RED-based, per-active flow approach has been proposed for providing fairness between flows [32]. 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 number of states 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, $O(N)$ states 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.

E. 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 IV-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 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.

VI. Conclusion and Future Work

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

REFERENCES

[1] S. Athuraliya, S. Low, V. Li, and Q. Yin, "REM active queue management," *IEEE Network Mag.*, vol. 15, pp. 48–53, May 2001.

[2] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, July 1970.

[3] 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, "Recommendations on queue management and congestion avoidance in the Internet,", RFC 2309, Apr 1998.

[4] K. Cho, "A framework for alternate queueing: Toward traffic management by PC-UNIX based routers," in *USENIX Annu. Tech. Conf.*, June 1998, pp. 247–258.

[5] I. Cidon, R. Guerin, and A. Khamisy, "On protective buffer policies," *IEEE/ACM Trans. Networking*, vol. 2, pp. 240–246, June 1994.

[6] S. Doran, "RED experience and differentiated queueing," presented at the North American Network Operators' Group (NANOG) Meeting, Dearborn, MI, June 1998.

[7] K. Fall and S. Floyd. (1997, Feb.) Router mechanisms to support end-to-end congestion control. [Online]. Available: ftp://ftp.ee.lbl.gov/papers/collapse.ps

[8] W. Feng, D. Kandlur, D. Saha, and K. G. Shin, "Techniques for eliminating packet loss in congested TCP/IP networks," Univ. Michigan, Ann Arbor, MI, Tech. Rep. UM CSE-TR-349-97, Oct. 1997.

[9] ——, "A self-configuring RED gateway," in *Proc. IEEE INFOCOM*, Mar. 1999, pp. 1320–1328.

[10] W. Feng, D. Kandlur, D. Saha, and K. G. Shin, "Blue: A new class of active queue management algorithms," Univ. Michigan, Ann Arbor, MI, Tech. Rep. UM CSE-TR-387-99, Apr. 1999.

[11] ——, "Stochastic fair BLUE: A queue management algorithm for enforcing fairness," in *Proc. IEEE INFOCOM*, Apr. 2001, pp. 1520–1529.

[12] S. Floyd, "TCP and explicit congestion notification," *Comput. Commun. Rev.*, vol. 24, no. 5, pp. 10–23, Oct. 1994.

[13] S. Floyd and V. Jacobson, "On traffic phase effects in packet-switched gateways," *Internetworking: Research and Experience*, vol. 3, no. 3, pp. 115–156, Sept. 1992.

[14] ——, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Networking*, vol. 1, pp. 397–413, Aug. 1993.

[15] ——, "Link-sharing and resource management models for packet networks," *IEEE/ACM Trans. Networking*, vol. 3, pp. 365–386, Aug. 1995.

[16] R. Guerin, S. Kamat, V. Peris, and R. Rajan, "Scalable QoS provision through buffer management," in *Proc. ACM SIGCOMM*, Sept. 1998, pp. 29–40.

[17] C. Hollot, V. Misra, D. Towsley, and W. Gong, "On designing improved controllers for aqm routers supporting TCP flows," in *Proc. IEEE INFOCOM*, Apr. 2001, pp. 1726–1734.

[18] *IEEE 802.11 Standard*, June 1997.

[19] V. Jacobson, "Congestion avoidance and control," in *Proc. ACM SIGCOMM*, Aug. 1988, pp. 314–329.

[20] S. Kunniyur and R. Srikant, "Analysis and design of an Adaptive Virtual Queue (AVQ) algorithm for active queue management," in *Proc. ACM SIGCOMM*, Aug. 2001, pp. 123–134.

[21] W. Leland, M. Taqqu, W. Willinger, and D. Wilson, "On the self-similar nature of ethernet traffic (extended version)," *IEEE/ACM Trans. Networking*, vol. 2, pp. 1–15, Feb. 1994.

[22] D. Lin and R. Morris, "Dynamics of random early detection," in *Proc. ACM SIGCOMM*, Sept. 1997, pp. 127–137.

[23] S. McCanne and S. Floyd. (1996) ns-LBNL Network Simulator. [Online]. Available: http://www-nrg.ee.lbl.gov/ns/

[24] P. McKenney, "Stochastic fairness queueing," in *Proc. IEEE INFOCOM*, Mar. 1990, pp. 733–740.

[25] R. Morris, "TCP behavior with many flows," in *Proc. IEEE Int. Conf. Network Protocols*, Oct. 1997, pp. 205–211.

[26] (1998). Netperf. [Online]. Available: http://www.netperf.org/

[27] T. Ott, T. Lakshman, and L. Gong, "SRED: Stabilized RED," in *Proc. IEEE INFOCOM*, Mar. 1999, pp. 1346–1355.

[28] V. Paxson, "End-to-end internet packet dynamics," in *Proc. ACM SIGCOMM*, Sept. 1997, pp. 139–152.

[29] V. Paxson and S. Floyd, "Wide-area traffic: The failure of Poisson modeling," in *Proc. ACM SIGCOMM*, Aug. 1994, pp. 257–268.

[30] K. K. Ramakrishan and R. Jain, "A binary feedback scheme for congestion avoidance in computer networks," *ACM Trans. Comput. Syst.*, vol. 8, no. 2, pp. 158–181, May 1990.

[31] K. Ramakrishnan and S. Floyd, "A proposal to add Explicit Congestion Notification (ECN) to IP,", RFC 2481, Jan. 1999.

[32] B. Suter, T. V. Lakshman, D. Stiliadis, and A. Choudhury, "Design considerations for supporting TCP with per-flow queueing," in *Proc. IEEE INFOCOM*, Mar. 1998, pp. 299–306.

[33] V. K. Garg, K. Smolik, and J. E. Wilkes, *Applications of CDMA in Wireless/Personal Communications*. Englewood Cliffs, NJ: Prentice-Hall, Oct. 1996.

[34] C. Villamizar and C. Song, "High-performance TCP in ANSNET," *Comput. Commun. Rev.*, vol. 24, no. 5, pp. 45–60, Oct. 1994.

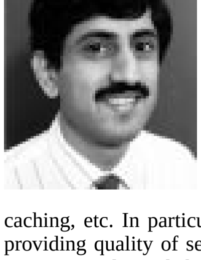
Wu-chang Feng received the B.S. degree in computer engineering from Penn State University, State College, and the M.S.E. and Ph.D. degrees in computer science engineering from the University of Michigan, Ann Arbor.

He is currently an Assistant Professor at the Oregon Graduate Institute (OGI) at the Oregon Health and Science University (OHSU) where he is currently a member of the Systems Software Laboratory. Prior to joining OGI/OHSU, he served as a Senior Architect at Proxinet/Pumatech, Inc.

Kang G. Shin (S'75–M'78–SM'83–F'92) received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea, in 1970 and the M.S. and Ph.D degrees in electrical engineering from Cornell University, Ithaca, NY, in 1976 and 1978, respectively.

He is the O'Connor Chair Professor of Computer Science, and the Founding Director of the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI. His current research focuses on QoS-sensitive networking and computing as well as on embedded real-time OS, middleware and applications, all with emphasis on timeliness and dependability. He has supervised the completion of 42 Ph.D. theses, and authored/coauthored over 600 technical papers and numerous book chapters in the areas of distributed real-time computing and control, computer networking, fault-tolerant computing, and intelligent manufacturing. He has coauthored (with C. M. Krishna) *Real-Time Systems* (New York: McGraw Hill, 1997).

Dr. Shin received the Outstanding IEEE TRANSACTIONS ON AUTOMATIC CONTROL Paper Award in 1987, and the Research Excellence Award in 1989, the Outstanding Achievement Award in 1999, the Service Excellence Award in 2000, and the Distinguished Faculty Achievement Award in 2001 from the University of Michigan. He has also coauthored papers with his students which have received the Best Student Paper Awards from the 1996 IEEE Real-Time Technology and Application Symposium and the 2000 UNSENIX Technical Conference.



Dilip D. Kandlur (S'90–M'91) received the M.S.E. and Ph.D. degrees in computer science and engineering from the University of Michigan, Ann Arbor.

He heads the Networking Software and Services Department at the IBM T. J. Watson Research Center, Yorktown Heights, NY. Since joining the IBM T. J. Watson Research Center, his research work has covered various aspects of providing quality of service in hosts and networks and their application to multimedia systems, network and server performance, web caching, etc. In particular, he has worked on protocols and architectures for providing quality of service in IP and ATM networks, their impact on transport protocols, and their realization in protocol stacks for large servers and switch/routers. He holds ten U.S. patents.

Dr. Kandlur is a member of the IEEE Computer Society and currently Vice-Chair of the IEEE Technical Committee on Computer Communications. He has been awarded an Outstanding Technical Achievement Award for his work in creating the QoS architecture for IBM server platforms, and has been recognized as an IBM Master Inventor.

Debanjan Saha (M'01) received the B.Tech. degree from the Indian Institute of Technology, India, and the M.S. and Ph.D. degrees from the University of Maryland at College Park, all in computer science.

He manages the advanced development group at Tellium, Inc., West Long Branch, NJ. Prior to his tenure at Tellium, he spent several years at IBM Research and Lucent Bell Labs, where he designed and developed protocols for IP routers and Internet servers. He is actively involved with various standards bodies, most notably IETF and OIF. He also serves as editor of international journals and magazines, technical committee members of workshops and conferences. He is a notable author of numerous technical articles on various topics of networking and is a frequent speaker at academic and industrial events.