BLUE: An Alternative Approach to Active Queue Management

Wu-chang Feng
Oregon Graduate Institute
wuchang@cse.ogi.edu

Dilip Kandlur
IBM Research
kandlur@us.ibm.com

Debanjan Saha
Tellium
dsaha@tellium.com

Kang G. Shin
Univ. of Michigan
kgshin@eecs.umich.edu

ABSTRACT

This paper exposes an inherent weakness in current active queue management techniques such as RED in that they rely on queue lengths to indicate 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 utilization to manage congestion. Using simulation and controlled experiments, BLUE is shown to significantly outperform RED in providing lower packet loss rates and smaller queuing delays to networked applications such as interactive audio and video.

1. INTRODUCTION

It is important to avoid high packet loss rates and high network delays 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 [13]. When high packet delays occur, the quality of interactive multimedia applications deteriorates significantly causing them to become unusable. In order to address these problems, the IETF is considering the deployment of explicit congestion notification (ECN) [11, 20, 19] along with active queue management techniques such as RED (Random Early Detection) [12, 3]. While ECN is necessary for eliminating packet loss and high queuing delays [9, 10], this paper shows that RED, even when used with ECN, is ineffective in preventing both.

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 and its variants [9, 17, 1] are certainly an improvement over traditional drop-tail queues, they all have a fundamental problem in that they rely on queue lengths as an estimator of congestion. While the presence of a persistent queue indicates congestion, its length gives very little information as to the severity of congestion, that is, the number of competing connections sharing the link. In a busy period, a single source transmitting at a rate greater than the bottleneck link capacity can cause a queue to build up just as easily as a large number of sources can. Since the RED algorithm relies on queue lengths, it has an inherent problem in determining the severity of congestion. As a result, RED requires a wide range of parameters to operate correctly under different congestion scenarios [14, 5]. While RED can achieve an ideal operating point, it can only do so when it has a large amount of buffer space and is correctly parameterized [7, 21].

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 an extremely small amount of buffer space.

NOSSDAV'01, June 25-26, 2001, Port Jefferson, New York, USA.

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

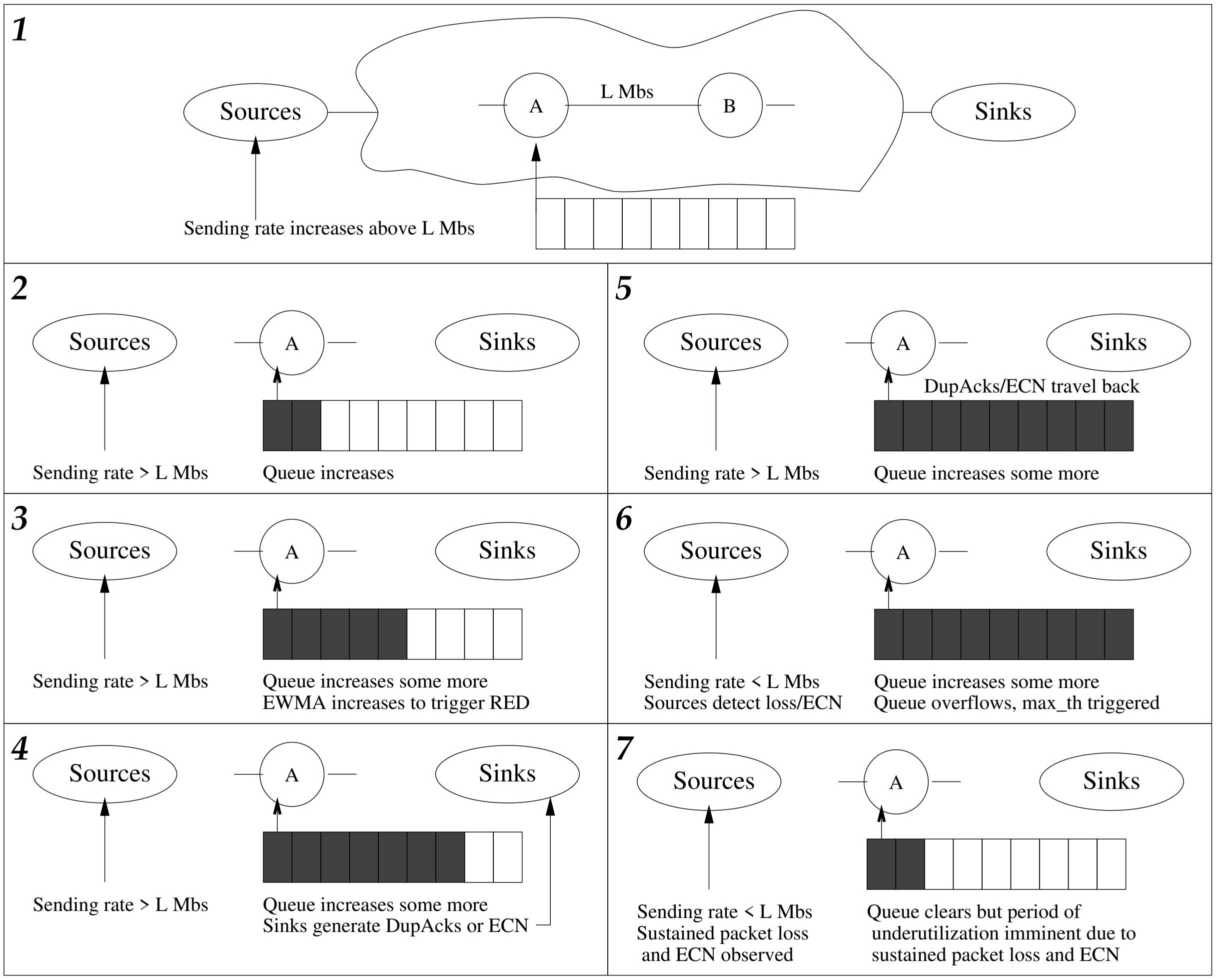


Figure 1: RED example

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 [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. 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 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 [9, 8] 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 [21]. 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.

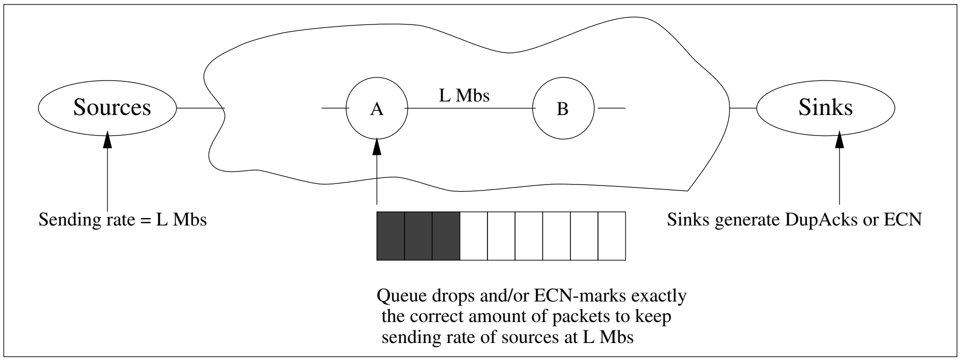


Figure 2: Ideal scenario

```
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

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

Table 1: RED configurations

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

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 overcome many of RED's shortcomings. In particular, the results show that BLUE converges to the ideal operating point shown in Figure 2 in 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. 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. In the figure, "now" corresponds to the current time while *last_update* corresponds to the last time p_m was changed. While the experiments in this chapter fix *freeze_time* as a constant, this value should be randomized in order to avoid global synchronization. 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.

While there are a myriad of ways in which p_m can be managed, it is important to point out that the selection of parameters directly determines how well the control system (BLUE and the aggregation of TCP sources) behaves. If p_m changes too quickly, as it does with RED, then periodic underutilization and packet loss can occur. If p_m changes too slowly, then p_m never matches the continually changing dynamics of the traffic aggregate that it is attempting to control. Recent work has shown that aggregate connection

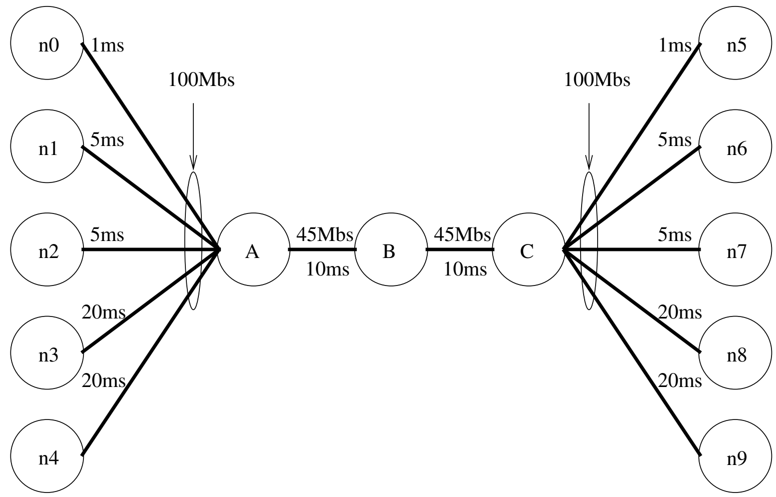


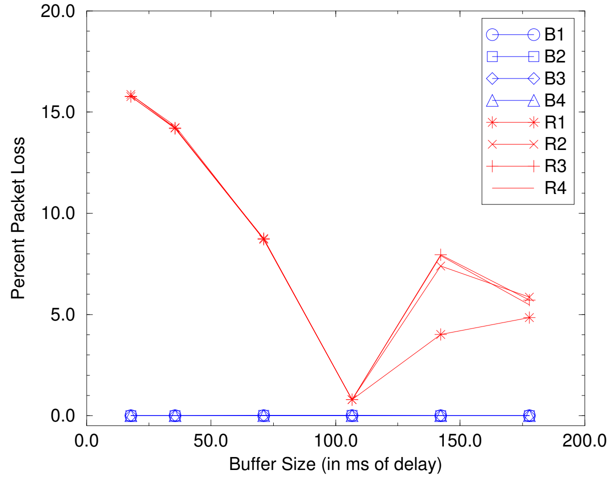
Figure 4: Network topology

behavior along with throughput and available bandwidth behavior exhibits high temporal correlation sometimes on the order of minutes and hours [2, 18]. Because of this, we follow two simple rules of thumb when selecting the parameters used for BLUE in this paper. The first is to select *freeze_time* on the order of the average round-trip times observed across the Internet. This allows any changes to p_m to take effect before additional changes are made. We then select values for $d1$ and $d2$ which allow p_m to cover th the entire range from 0 to 1 (or 1 to 0) on the order of several seconds so that the algorithm can adapt to sudden changes in load. While the experiments in this paper study a small range of parameter settings, experiments with additional parameter settings which follow the above rule of thumb show similar results 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.

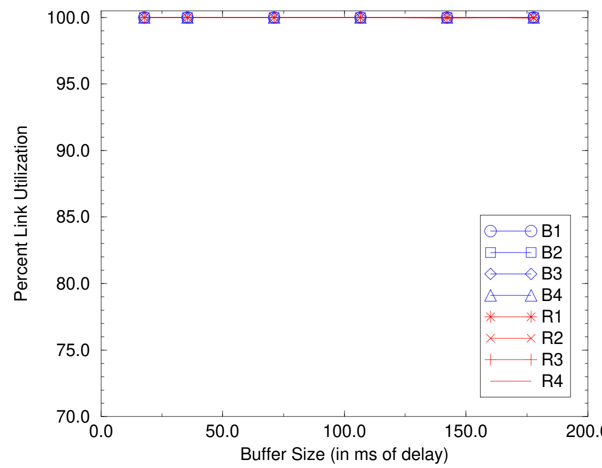
3.2 Packet loss rates using RED and BLUE

In order to evaluate the performance of BLUE, a number of experiments were run using **ns** [15] 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 [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 algorithm becomes decoupled from the queue length and thus acts more like BLUE. Table 1 shows the configurations used for RED and Table 2 shows the configurations used for BLUE. 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 $100ms$ corresponding to typical round-trip delays observed across the Internet.

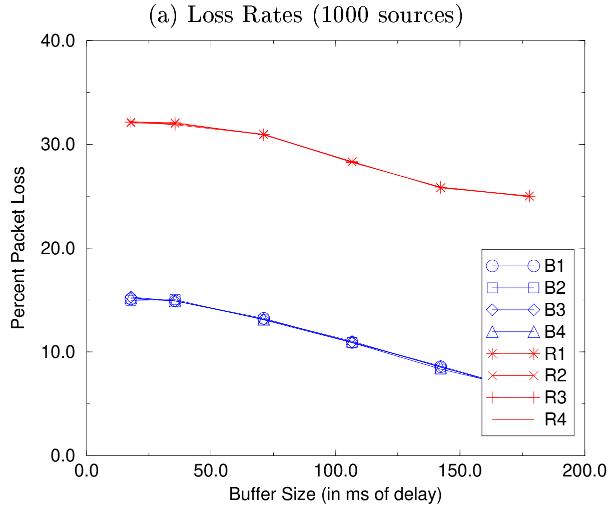
Figure 5 shows the loss rates observed over different queue sizes using both BLUE and RED with 1000 and 4000 connections present. In these experiments, the queue at the bottleneck link between A and B is sized from $100KB$ to $1000KB$. This corresponds to queueing delays which range from $17.8ms$ and $178ms$ 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. 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



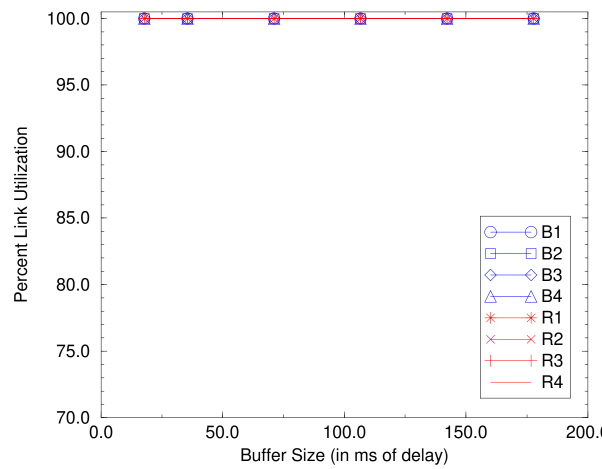
(a) Loss Rates (1000 sources)



(b) Link Utilization (1000 sources)



(c) Loss Rates (4000 sources)



(d) Link Utilization (4000 sources)

Figure 5: Performance of Red and Blue

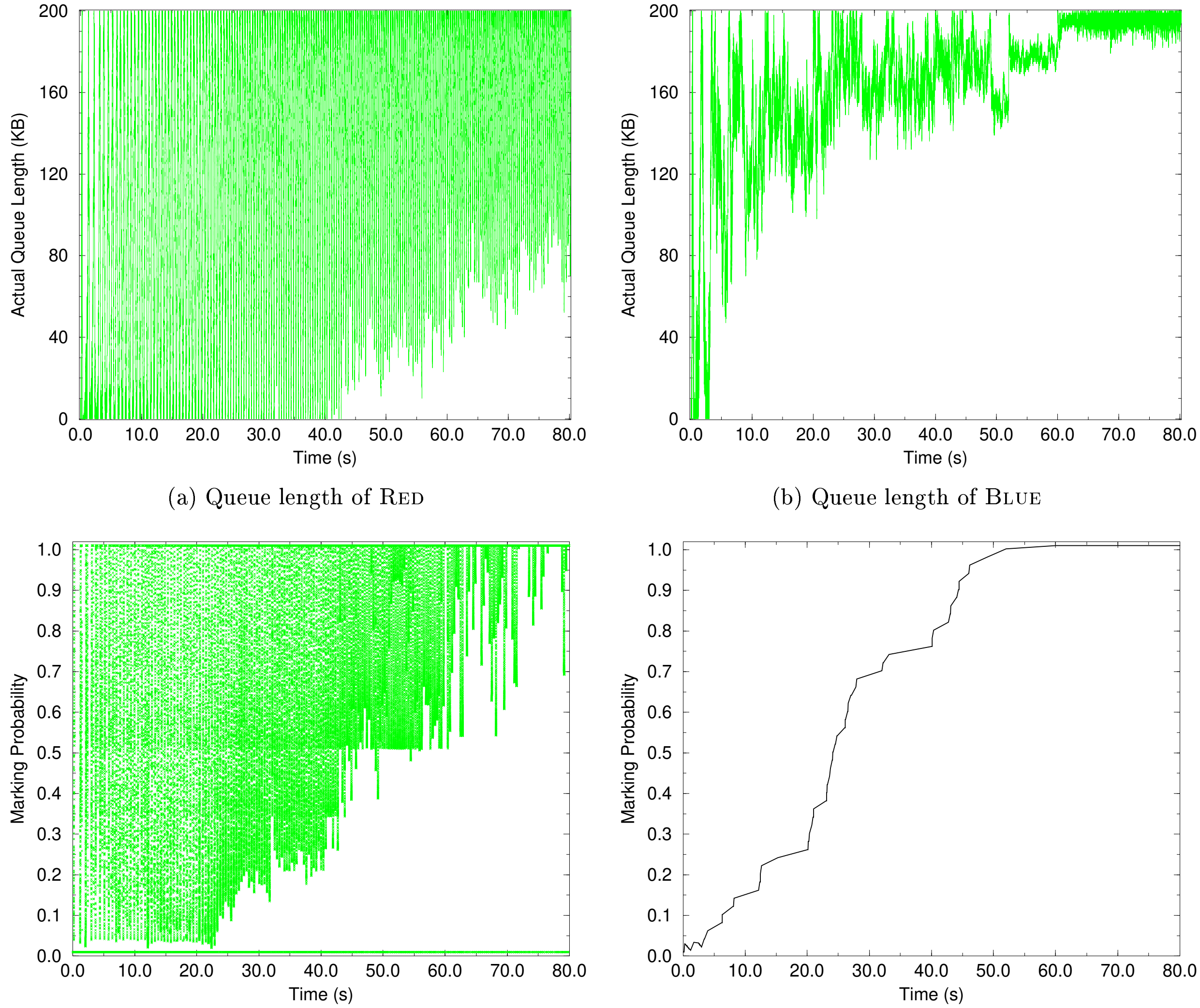
a result, packet loss is often accompanied with deterministic marking. When combined, this leads again to fluctuations in queue length. At a load which is perfectly selected, the average queue length of Red can remain at max_{th} and the queue can avoid packet loss and prevent queue fluctuations by marking 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. The figure also shows that in order for Red to match Blue's observed loss rates using $17.8ms$ ($100KB$) of buffer space, Red requires at least 6 times more buffer space, adding considerable amount of queueing delay.

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 and the ability to run interactive applications over the Internet. In addition, smaller buffering requirements allow more memory to be allocated to high priority packets [6], 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, Figures 6(a) and 6(b) compare 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 in-



(a) Queue length of Red

(b) Queue length of Blue

(c) Marking probability of Red ($\frac{p_b}{1-count \times p_b}$)

(d) Marking probability of Blue (p_m)

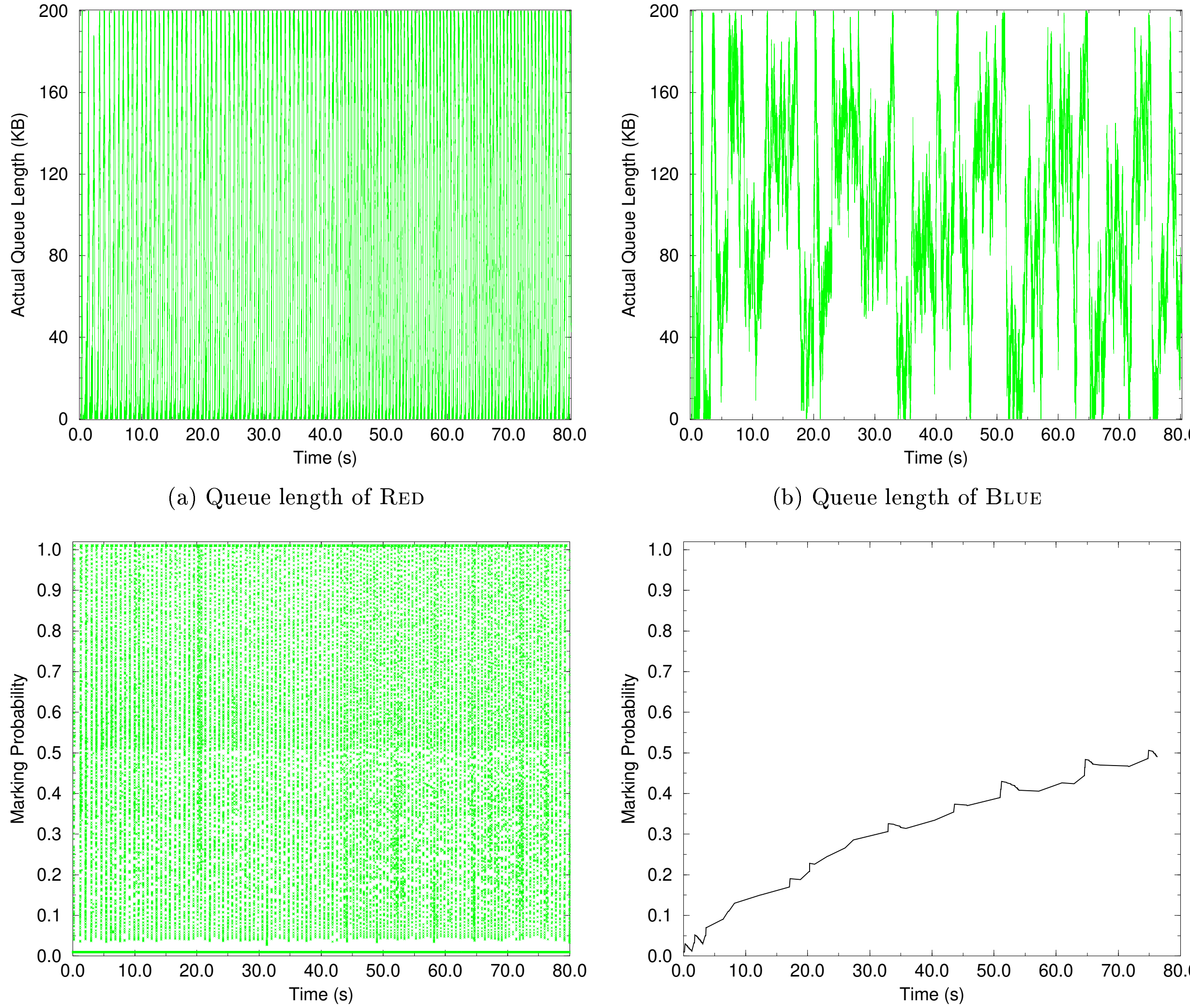
Figure 6: Queue lengths and marking probabilities of Red and Blue

creasing 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 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 6(c) plots the the marking probability of Red throughout the experiment ($\frac{p_b}{1-count \times p_b}$). Note that 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 6(c) fluctuates considerably. In contrast, Figure 6(d) 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 $t = 60s$ 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.

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



(a) Queue length of RED

(b) Queue length of BLUE

(c) Marking probability of RED ($\frac{p_b}{1-count \times p_b}$)

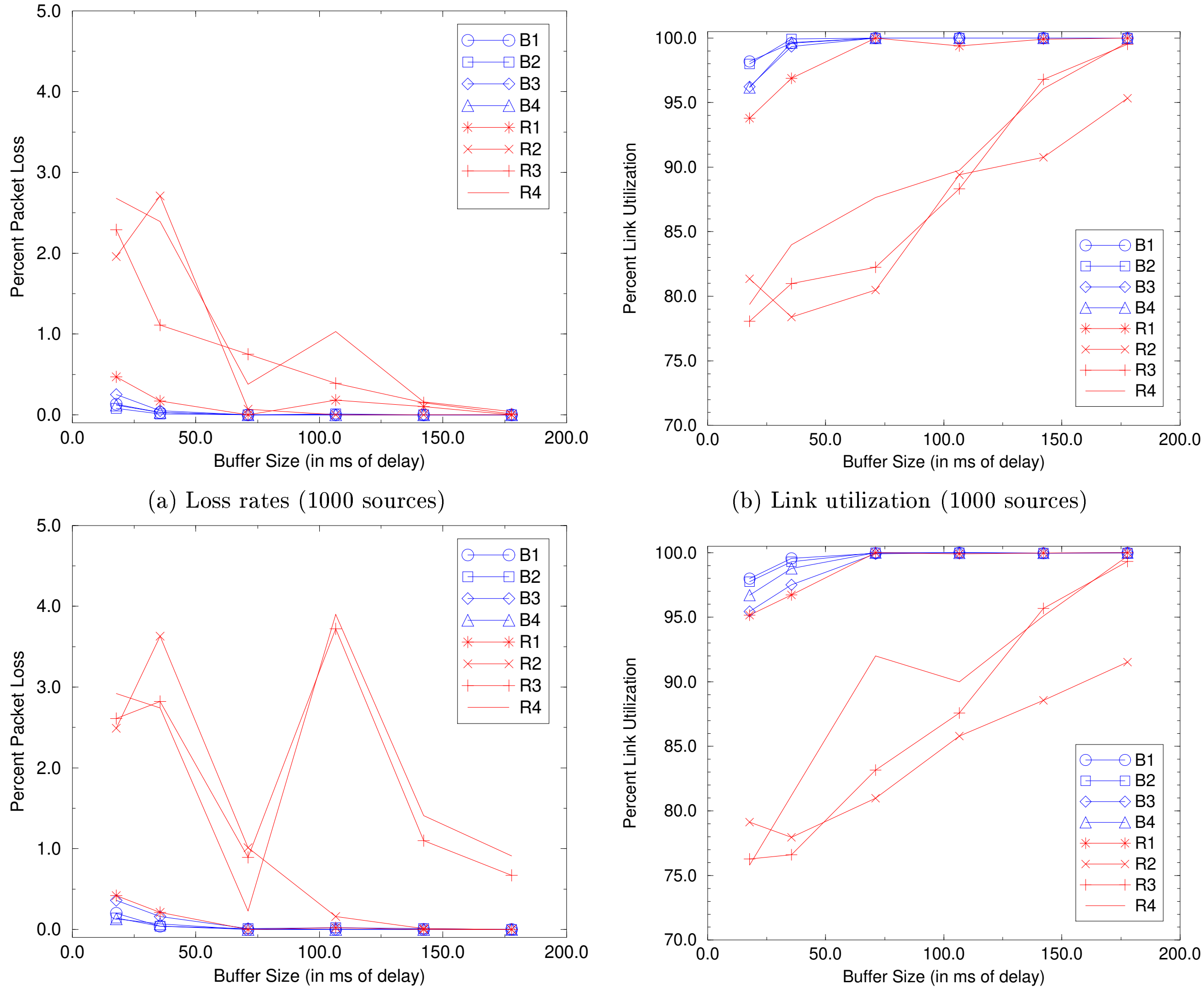
(d) Marking probability of BLUE (p_m)

Figure 7: Queue lengths and marking probabilities of RED and BLUE with ECN timeouts

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. Figures 7(a) and 7(b) show the queue length plots of RED and BLUE using the same experiments as in Section 3.2 with TCP sources enabled with ECN timeouts. Figure 7(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 7(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.

Figures 7(c) and 7(d) show 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 8 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 compar-



(a) Loss rates (1000 sources) (b) Link utilization (1000 sources)

(c) Loss rates (4000 sources) (d) Link utilization (4000 sources)

Figure 8: Performance of RED and BLUE with ECN timeouts

ison 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 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 [4]. In this implementation, ECN uses two bits of the type-of-service (ToS) field in the IP header [20]. 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 9. 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` [16] 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

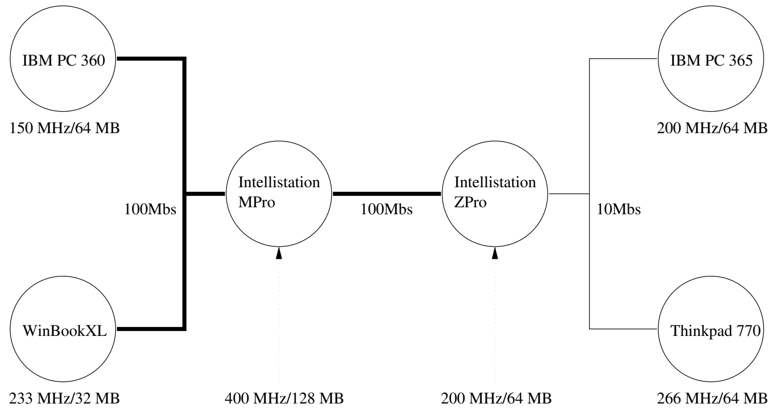
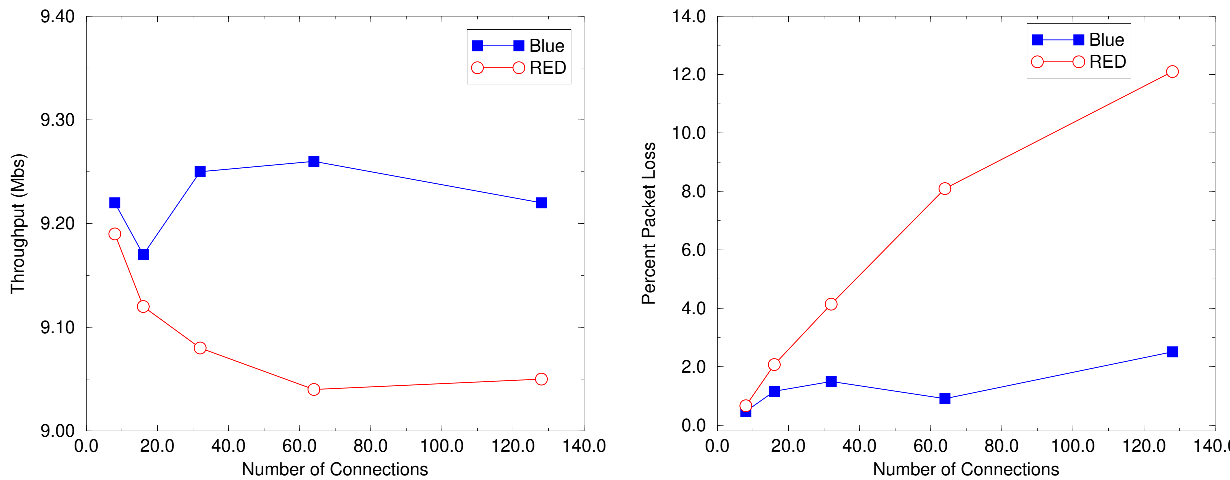


Figure 9: Experimental testbed



(a) Throughput

(b) Percent packet loss

Figure 10: Queue management performance

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 10(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-second intervals and then averaged. Note that the TCP sources used in the experiment do not implement ECN timeouts. As Figure 10(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 10(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.

4. CONCLUSION

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. By using packet loss and link utilization history of the congested queue, rather than queue lengths, BLUE significantly outperforms queue management algorithms based on RED in terms of packet loss and link utilization. More importantly, BLUE significantly reduces the buffering requirements at congested routers, thus reducing the observed network delays which is critical for interactive audio and video applications.

5. REFERENCES

[1] F. Anjum and L. Tassiulas. An Algorithm to Achieve Fairness in the Internet. University of Maryland Techreport TR_99-17, 1999.

[2] H. Balakrishnan, V. Padmanabhan, S. Seshan, M. Stemm, and R. Katz. TCP Behavior of a Busy Internet Server: Analysis and Improvements. In *Proc. IEEE INFOCOM*, March 1998.

[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*, April 1998.

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

[5] M. Christiansen, K. Jeffay, D. Ott, and F. Smith. Tuning RED for Web Traffic. In *Proceedings of ACM SIGCOMM*, September 2000.

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

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

[8] 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.

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

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

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

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

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

[14] M. May, J. Bolot, C. Diot, and B. Lyles. Reasons Not to Deploy RED. In *IWQoS*, March 1999.

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

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

[17] T. Ott, T. Lakshman, and L. Wong. SRED: Stabilized RED. In *Proc. IEEE INFOCOM*, March 1999.

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

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

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

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