Queue Management for Short-Lived TCP Flows in Backbone Routers

Anshul Kantawala and Jonathan Turner
Department of Computer Science
Washington University
St. Louis, MO 63130
{*anshul,jst*}*@arl.wustl.edu*

Abstract—**Packets in the Internet can experience large queueing delays during busy periods. Backbone routers are generally engineered to have large buffers, in which packets may wait as long as half a second (assuming FIFO service, longer otherwise). During congestion periods, these buffers may stay close to full, subjecting packets to long delays, even when the intrinsic latency of the path is relatively small. This paper studies the performance improvements that can be obtained for short-lived TCP flows by using more sophisticated packet schedulers, than are typical of Internet routers. The results show that the large buffers found in WAN routers contribute only marginally to improving router throughput, and the higher delays that come with large buffers makes them a dubious investment. The results also show that better packet scheduling algorithms can produce dramatic improvements in fairness. Using ns-2 simulations, we show that algorithms using multiple queues can significantly outperform RED and Blue, especially at smaller buffer sizes. Given a traffic mix of short-lived TCP flows with different round-trip times, longer round-trip time flows achieve 80% of their fair-share using multiqueue schedulers, compared to 40% under RED and Blue. We observe a similar performance improvement for multi-hop paths. We also show that performance results can be reliably scaled across a wide range of parameter values, so long as the ratio of the buffer size to the link bandwidth-delay product is held invariant.**

I. Introduction

Backbone routers in the Internet are typically configured with buffers that are several times times larger than the product of the link bandwidth and the typical round-trip delay on long network paths. Such buffers can delay packets for as much as half a second during congestion periods. When such large queues carry heavy TCP traffic loads, and are serviced using the Tail Drop policy, the large queues remain close to full most of the time. Thus, even if each TCP flow is able to obtain its share of the link bandwidth, the end-to-end delay remains very high. This is exacerbated for flows with multiple hops, since packets may experience high queueing delays at each hop. This phenomenon is well-known and has been discussed by Hashem [1] and Morris [2], among others.

To address this issue, researchers have developed alternative queueing algorithms which try to keep average queue sizes low, while still providing high throughput and link utilization. The most popular of these is *Random Early Discard* or RED [3]. RED maintains an exponentially-weighted moving average of the queue length which is used to detect congestion. To make it operate it robustly under widely varying conditions, one must either dynamically adjust the parameters or operate using relatively large buffer sizes [4], [5]. Recently another queueing algorithm called Blue [6], was proposed to improve upon RED. Blue adjusts its parameters automatically in response to queue overflow and underflow events. Although Blue does improve over RED in certain scenarios, its parameters are also sensitive to different congestion conditions and network topologies.

In this paper, we investigate how packet schedulers using multiple queues can improve performance over existing methods. Our goal is to find schedulers that satisfy the following objectives:

- *High throughput when buffers are small*. This allows queueing delays to be kept low.
- *Insensitivity to operating conditions and traffic*. This reduces the need to tune parameters, or compromise on performance.
- *Fair treatment of different flows*. This should hold regardless of differences in round-trip delay or number of hops traversed.

The results presented here show that both RED and Blue are deficient in these respects. Both perform fairly poorly when buffer space is limited to a small fraction of the round-trip delay.

In a previous study [9], we investigated the performance of multiple queues using long-lived TCP flow traffic. The focus of this paper is studying the performance of multiple queues using short-lived TCP flow traffic. A majority of the Internet traffic today is HTTP (web-traffic), which consists of short-lived TCP flows transferring web pages. Even with p-HTTP (persistent HTTP) connections, the resulting TCP traffic is bursty (downloading a web page) with long idle periods (user pause between web page downloads) and can be emulated by multiple short-lived TCP flows.

Another regularly observed phenomenon for queues with Tail Drop is big swings in the occupancy of the bottleneck link queue. One of the main causes for this is the synchronization of TCP sources going through the bottleneck link. Although RED and Blue try to alleviate the synchronization problem by using a random drop policy, they do not perform well with buffers which are a fraction of the bandwidth-delay product. When buffers are very small, even with a random drop policy, there is a high probability that all flows suffer a packet loss. However, with per-flow queueing, we can explicity control the number of flows that suffer a packet loss and thus significantly reduce synchronization among flows.

One of the limitations of evaluating algorithms using simulations is that we cannot replicate the scale of real networks. Thus, a common question is whether the results obtained will hold when the number of sources and/or bottleneck bandwidth is increased by one or more orders of magnitude. In this paper, we address the issue as to how the average goodput of TCP flows is affected by changes in network parameters, namely number

This work is supported in part by NSF Grant ANI-9714698

of sources, bottleneck link bandwidth and RTTs and show that these parameter values can have a wide range of values without a big impact on performance.

The rest of the paper is organized as follows. Section 2 describes the new multi-queue methods investigated here. Section 3 documents the configurations used for the simulations and the parameters used for evaluating our algorithms. Section 4 compares the performance results of the proposed multi-queue methods against RED, Blue and Tail Drop. Section 5 discusses the sensitivity of TCP performance to changes in network parameters and Section 6 concludes the paper.

II. Algorithms

Given the problems with existing congestion buffer management algorithms, we decided to evaluate a fair queueing discipline for managing TCP flows. We started with using Deficit Round Robin (DRR) [7]. DRR is an approximate fair-queueing algorithm that requires only $O(1)$ work to process a packet and thus it is simple enough to be implemented in hardware. Also, since there are no parameters to set or fine tune, it makes it usable across varying traffic patterns. We evaluated three different packet-discard policies.

1. **DRR with Longest Queue Drop**

Our first policy combined DRR with packet-discard from the longest active queue. For the rest of the paper, we refer to this policy as plain DRR or DRR, since this packet-discard policy is part of the original DRR algorithm [7] and was first proposed by McKenney in [8]. In [9], we present more details and motivation for developing the DRR variations presented below.

2. **Throughput DRR (TDRR)**

In this algorithm, we store a throughput value associated with each DRR queue. The throughput parameter is maintained as an exponentially weighted average and is used in choosing the drop queue. The exponential weight used in our simulations is 0.03125. We found that TDRR is not very sensitive to the weight parameter and performed equally well for weights ranging from 0.5 to $1.0e-6$. The discard policy for a new packet arrival when the link buffer is full, is to choose the queue with the highest throughput (amongst the currently active DRR queues) to drop a packet. Intuitively, this algorithm should penalize higher throughput TCP flows more and thus achieve better fairness and our simulation results do confirm this.

3. **Queue State DRR (QSDRR)**

Since TDRR has an overhead associated with computing and storing a weighted throughput value for each DRR queue, we investigate another packet-discard policy which adds some hysteresis to plain DRR's longest queue drop policy. The idea is that once we drop a packet from one queue, we keep dropping from the same queue when faced with congestion until that queue is the smallest amongst all active queues. This policy reduces the number of flows that are affected when a link becomes congested. This reduces the TCP synchronization effect and reduces the magnitude of the resulting queue length variations. A detailed description of this algorithm is presented in Figure 1.

```
Let Q be a state variable which is
  undefined initially.
if Q is not defined
  Set Q to current longest queue
else (Q is defined)
  if Q is shorter than all active queues
    Set Q to current longest queue
Drop packets from front of Q
```

Fig. 1. Algorithm for QSDRR

III. Simulation Environment

In order to evaluate the performance of DRR, TDRR and QSDRR, we ran a number of experiments using ns-2. In this paper, we investigate the performance of our algorithms for short-lived TCP connections. In our simulations, we emulate short-lived TCP flows using on-off TCP sources. The *on-phase* models an active TCP flow sending data, while the *off-phase* models the inter-arrival time between connections. To effectively compare the times taken to service each burst under different algorithms, we fix the data transferred per connection (during *on-phase*) to 256 packets (384KB). The idle time between bursts is exponentially distributed with a mean of 2 seconds.

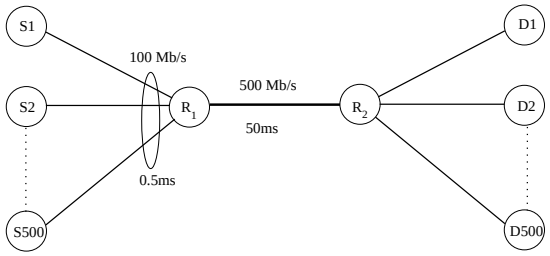


Fig. 2. Single Bottleneck Link Network Configuration

We compared the performance over a varied set of network configurations and traffic mixes which are described below. In all our experiments, we used TCP sources with 1500 byte packets and the data collected is over a 100 second simulation interval. We ran experiments using TCP Reno and TCP Tahoe and obtained similar results for both; hence, we only show the results using TCP Reno sources. For each of the configurations, we varied the bottleneck queue size from a 100 packets to 20,000 packets. 20,000 packets represents a half-second buffer which is a common buffer size deployed in current commercial routers. We ran several simulations to determine the best parameter values for RED and Blue for our simulation environment, to ensure a fair comparison against our multi-queue based algorithms.

A. Single Bottleneck Link

The network configuration for this set of experiments is shown in Figure 2. $\{S_1, S_2, ...S_{500}\}$ are the TCP sources, each connected by 100Mb/s links to the bottleneck link. The destinations, named $\{D_1, D_2, ...D_{500}\}$, are directly connected to the router R_2. All 500 TCP sources are started simultaneously to simulate a worst-case scenario whereby TCP sources are synchronized in the network.

B. Multiple Roundtrip-time Configuration

The network configuration for this set of experiments is shown in Figure 3. This configuration is used to evaluate the

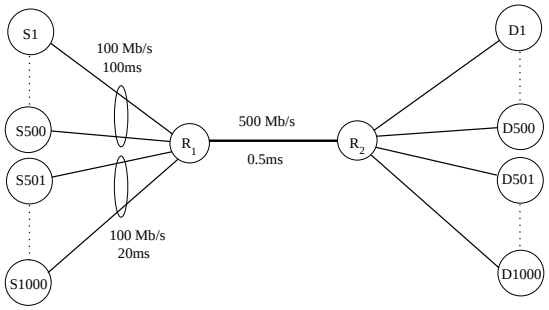


Fig. 3. Multiple Roundtrip-time Network Configuration

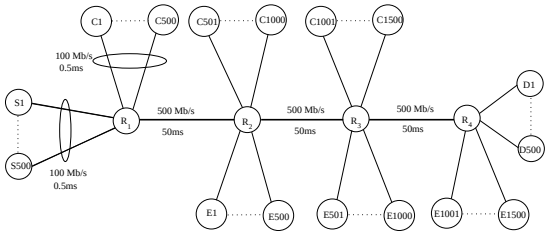
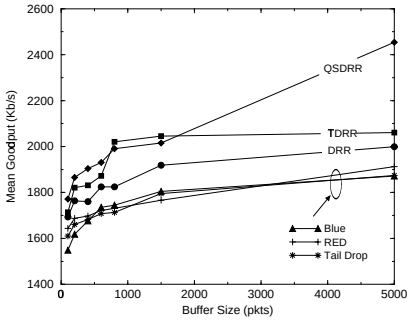
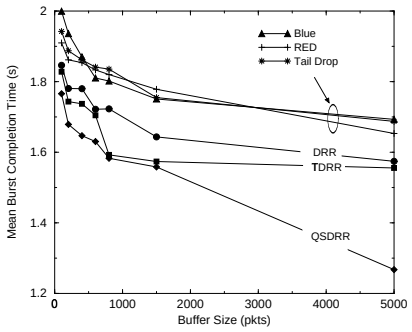


Fig. 4. Multi-Hop Path Network Configuration



(a) Mean Goodput



(b) Mean Burst Completion Time

Fig. 5. Performance of short burst TCP flows over a single bottleneck link

performance of the different queue management policies given two sets of TCP flows with widely varying roundtrip-times over the same bottleneck link. The source connection setup is similar to the single-bottleneck configuration, except for the access link delays for each source and the total number of sources. We simulated 1000 TCP sources, 500 sources with link delay set to 20ms, and 500 sources with link delay set to 100ms.

C. Multi-Hop Path Configuration

The network configuration for this set of experiments is shown in Figure 4. In this configuration, we have 500 TCP sources traversing three bottleneck links and terminating at R_3. In addition, on each link, there are 500 TCP sources acting as cross-traffic. We use this configuration to evaluate the performance of the different queue management policies for multi-hop TCP flows competing with shorter one-hop cross-traffic flows.

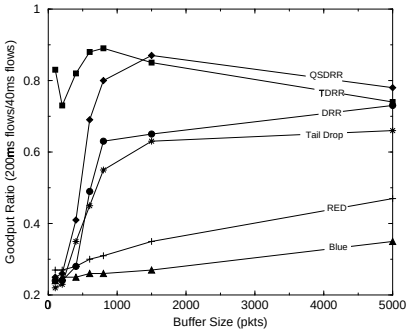
IV. RESULTS

Figure 5(a) shows the mean goodput achieved by the TCP flows and Figure 5(b) shows the mean burst completion times for the flows. We notice that Blue, RED and Tail Drop have almost exactly the same performance in terms of mean goodput achieved and burst completion times for all buffer sizes, whereas the DRR schemes are uniformly better. For buffer sizes less than 2000 packets, TDRR and QSDRR exhibit about 10% better goodput performance over Blue, RED and Tail Drop. However, it is interesting to note that QSDRR is almost 30% better than the non-DRR policies at a buffer size of 5000 packets. The results are similar for the burst completion times.

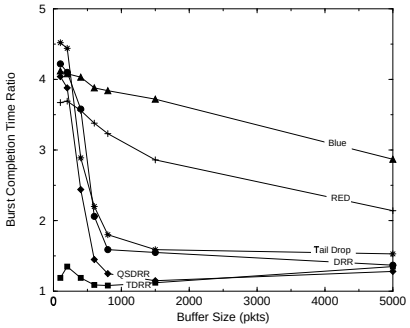
Figure 6(a) shows the ratios of the goodputs obtained by 200ms round-trip time flows over the goodputs of the 40ms round-trip time flows. For this configuration, for buffer sizes less than a 1000 packets, QSDRR and TDRR outperform Blue and RED by more than 100%. The ratio of goodputs is used to illustrate the fairness of each algorithm. The closer the ratio is to one, the better the algorithm is in delivering fair-share to different round-trip time flows. In this case, even Tail Drop performs significantly better than Blue and RED, showing that for short-lived flows with different round-trip times, Blue and RED cannot deliver good fair-sharing of the bottleneck bandwidth. Figure 6(b) shows the ratios of burst completion times of the 200ms round-trip time flows over the 40ms round-trip time flows. In this case, QSDRR and TDRR remain close to one (which is the ideal fairness), whereas Blue has the worst performance, with the 200ms round-trip time flows taking almost *three times* the time to complete a burst compared to the 40ms round-trip time flows, even for 5000 packet buffers.

Figure 7(a) shows the ratios of the goodputs achieved by the end-to-end flows over the cross-traffic flows. In this configuration, we see that the non-DRR policies perform very poorly, allowing the end-to-end flows a mere 30% of the goodput achieved by the cross-traffic flows. On the other hand, QSDRR and even DRR outperform the non-DRR schemes by 40% for buffer sizes less than 2000 packets. QSDRR is almost *2 times* better than the non-DRR policies for a buffer size of 5000 packets. Since TDRR maintains an exponentially weighted throughput average for a fairly long period, it outperforms all policies. For short-lived TCP flows, DRR and QSDRR cannot deliver the same fairness as TDRR since they do not maintain long-term state.

Figure 7(b) shows the ratios of burst completion times of the end-to-end flows over the cross-traffic flows. Only TDRR can

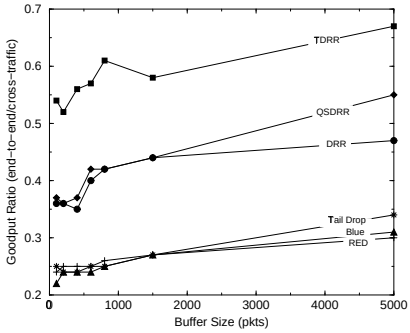


(a) Goodput Ratios (200ms flows/40ms flows)

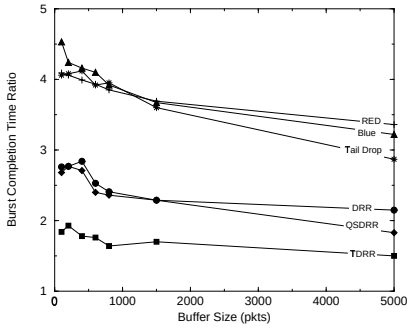


(b) Burst Completion Time Ratios (200ms flows/40ms flows)

Fig. 6. Performance of short burst TCP flows over a multiple round-trip time configuration



(a) Goodput Ratios (end-to-end flows/cross-traffic flows)



(b) Burst Completion Time Ratios (end-to-end flows/cross-traffic flows)

Fig. 7. Performance of short burst TCP flows over a multi-hop path configuration

achieve a ratio close to one, since it maintains long-term state. However, QSDRR and DRR perform reasonably well and beat the non-DRR policies by at least a factor of two. Even though the end-to-end traffic flows over three bottleneck links compared to just one bottleneck-link for the cross-traffic flows, QSDRR and DRR are able to achieve a burst completion time ratio of under two for a buffer size of 5000 packets. At the same buffer size, the non-DRR policies achieve fairly poor ratios ranging from 3.5 to 4.0.

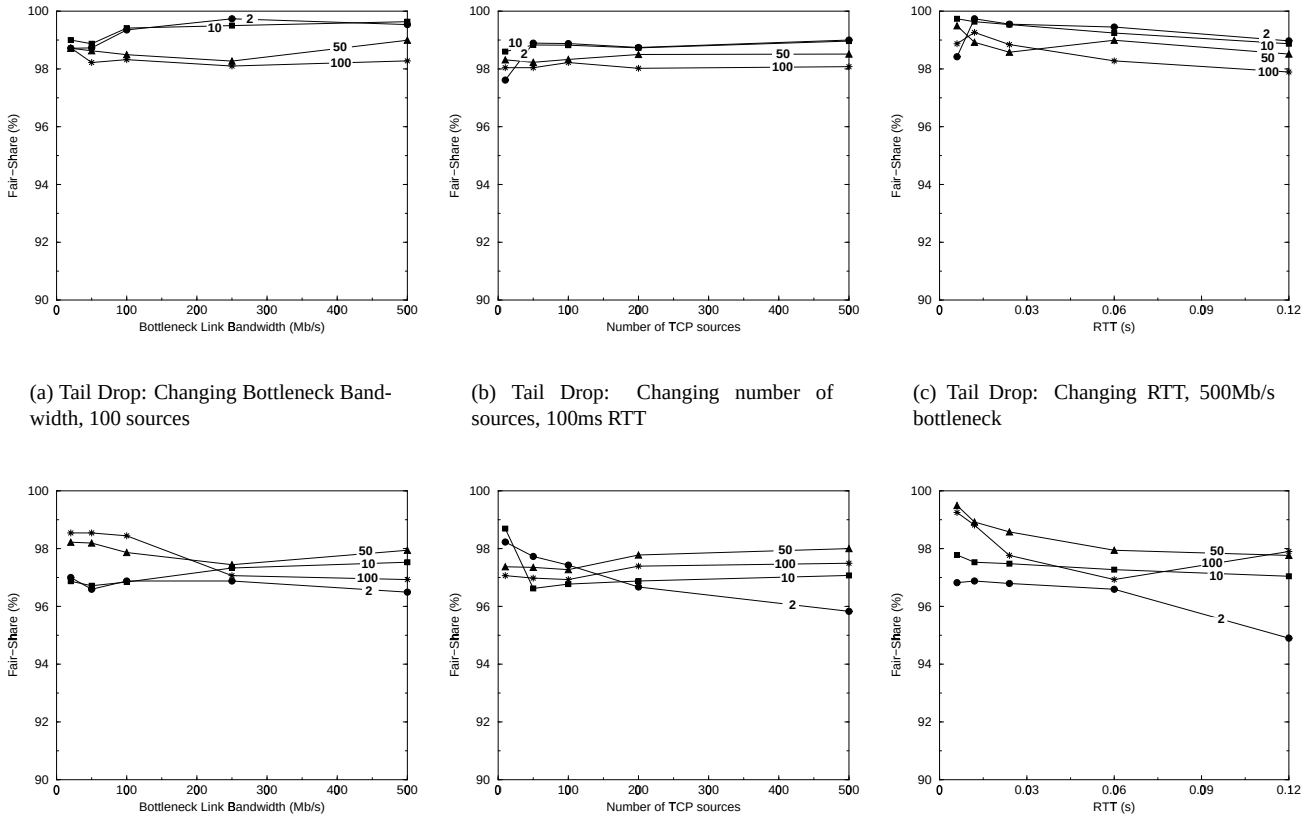
V. Usability over Large Networks

A common question is that though the simulation results show good performace for our configuration (1000 sources and 500Mb/s bottleneck link), will we still achieve the same performance when we have 20,000 sources over a 10Gb/s link? Realistically, it is impractical to simulate such large network configurations. However, we attempt to address this issue by showing that, if the ratio of the buffer size to the link bandwidth-delay product is held invariant, the performance is fairly insensitive to a wide range of changes in parameters such as number of sources, RTT and bottleneck link capacities. In our study, we varied the number of sources from 10 to a 1000, RTT times from 6ms to 1.5s and bottleneck link bandwidths from 20Mb/s to 3Gb/s.

A. Simulation Setup

We use the single bottleneck link configuration as shown in Figure 2 as our base topology. For each set of experiments (graphs), we evaluate four different *fair-share window sizes* (2, 10, 50, 100) for a TCP source. We define *fair-share window size* as the fair-share bandwidth per TCP flow times the RTT. The bottleneck link buffer is set to the bandwidth-delay product of the network configuration. The three different simulation scenarios we study are outlined below.

1. **Varying bottleneck link bandwidth**
For this experiment, we set the number of sources to 100 and vary the bottleneck link bandwidth from 20Mb/s to 500Mb/s. The RTT is scaled along with the bottleneck bandwidth to maintain the constant fair-share window size.
2. **Varying number of sources**
For this experiment, we set the RTT to 100ms and vary the number of sources from 20 to 500. The bottleneck link bandwidth is scaled along with the number of sources to maintain the constant fair-share window size.
3. **Varying RTT**
For this experiment, we set the bottleneck link bandwidth to 500Mb/s and vary the RTT from 6ms to 120ms. The number of sources are scaled along with the RTT to maintain the constant fair-share window size.



(a) Tail Drop: Changing Bottleneck Bandwidth, 100 sources

(b) Tail Drop: Changing number of sources, 100ms RTT

(c) Tail Drop: Changing RTT, 500Mb/s bottleneck

(d) QSDRR: Changing Bottleneck Bandwidth, 100 sources

(e) QSDRR: Changing number of sources, 100ms RTT

(f) QSDRR: Changing RTT, 500Mb/s bottleneck

Fig. 8. Performance of TCP flows over single bottleneck link while varying traffic mix and network parameters

B. Results

Figure 8(a), 8(b) and 8(c) show the effect on the mean fair-share goodput percentage achieved by the TCP flows using Tail Drop policy and Figure 8(d), 8(e) and 8(f) show the effect on TCP flows using QSDRR policy. From the above graphs, it is clear that changing bottleneck-link bandwidths, RTTs, number of sources or fair-share window sizes has a negligible impact on TCP goodputs over a congested link. This tells us that we can reliably apply results from a smaller scale simulation to a larger scale scenario, assuming that the simulations are for similar fair-share bandwidths.

VI. Conclusion

This paper has demonstrated the inherent weaknesses in current queue management policies commonly used in Internet routers. These weaknesses include limited ability to perform well under a variety of network configurations and traffic conditions, inability to provide a fair-sharing among competing TCP connections with different RTTs and relatively low link utilization and goodput in routers that have small buffers. In order to address these issues, we presented TDRR and QSDRR, two different packet-discard policies used in conjunction with a simple, fair-queueing scheduler, DRR. Through extensive simulations, we showed that TDRR and QSDRR significantly outperform RED and Blue for various configurations and traffic mixes in both the average goodput for each flow and the variance in goodputs. For very small buffer sizes, in the order of $5-10\%$ of the bandwidth-delay product, we showed that not only did our policies significantly outperform RED, Blue and Tail Drop, but that they were able to achieve near optimal goodput and fairness. We also showed that performance is insensitive to wide ranging changes in network parameters.

References

[1] E. Hashem, "Analysis of random drop for gateway congestion control", Tech. Rep. LCS TR-465, Laboratory for Computer Science, MIT, 1989.

[2] Robert Morris, "Scalable TCP Congestion Control", in *IEEE INFOCOM 2000*, March 2000.

[3] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[4] S. Doran, "RED Experience and Differential Queueing", Nanog Meeting, June 1998.

[5] C. Villamizar and C. Song, "High Performance TCP in ANSNET", *Computer Communication Review*, vol. 24, no. 5, pp. 45–60, Oct. 1994.

[6] W. Feng, D. Kandlur, D. Saha, and K. Shin, "Blue: A New Class of Active Queue Management Algorithms", Tech. Rep. CSE-TR-387-99, University of Michigan, Apr. 1999.

[7] M. Shreedhar and George Varghese, "Efficient Fair Queueing using Deficit Round Robin", in *ACM SIGCOMM '95*, Aug. 1995.

[8] P. McKenney, "Stochastic Fairness Queueing", *Internetworking: Research and Experience*, vol. 2, pp. 113–131, Jan. 1991.

[9] Anshul Kantawala and Jonathan Turner, "Efficient Queue Management of TCP Flows", Tech. Rep. WUCS-01-22, Washington University, Aug. 2001.