RED in a Different Light (Updates)

Van Jacobson
Kathleen Nichols
Kedarnath Poduri

This document supplements the 1999 draft with material added in Februrary 28, 2004 to fill in some TBAs and provide more results on how we were looking at drops, delay, and other metrics. Most of the sections that overlap with the 1999 version have been removed.

Abstract

Packet networks require queues (buffers) to absorb short term arrival rate fluctuations. Yet network implementors have always observed that queues at bottlenecks tend to fill and stay filled, which contributes excessive delay and removes the ability to absorb bursts. In [1] Floyd and Jacobson proposed the RED (Random Early Detection) active queue management algorithm. RED is simple, robust and quite effective at reducing persistent queues. However, while it has been used widely and successfully on Internet routers, [1] offers little guidance on how to set configuration parameters and RED has gained the reputation of being very difficult to tune.

This paper develops RED in different way, treating it as a servo control loop and deriving all the loop parameters from measurable properties of a router. The result is a 'self-tuning' RED whose parameters are completely determined by the queue output bandwidth (average departure rate). This new RED performs substantially better than the original version and works for a much wider variety of traffic and link bandwidths. It also admits a substantially simpler and more efficient implementation, one particularly well suited for ASIC forwarding engines. Further, we believe our development explains the reasons for some of the problems others have noted with configuring the 93 RED

1 Introduction

Although queues and buffers are essential to the efficient operation of packet networks, queues at a bottleneck tend to fill up and remain full, adding unnecessary delay to traffic and losing the ability to absorb bursts. The first author has been interested in solving this problem for some time, originally proposing a Random Drop statistical congestion control algorithm in 1989 [13], a version of which was evaluated in [14] and not considered to be compelling. Subsequently, this was refined into RED (Random Early Detection) [1], considered one of the leading active queue managment candidates. The IRTF has urged the deployment of active queue management in the Internet [2].

The paper defining RED was first published in 1993 and there have since been a number of implementations, variations, imitations, and reports on its use [4][5]. The 1993 paper describes active queue management through randomly dropping (or marking)[1] arriving packets when the average queue length indicates congestion and makes a case for the efficacy of such an approach. Briefly, RED works by keeping a running estimate (using an exponential weighted moving average or EWMA) of the average queue size. When that size passes a lower threshold, the algorithm drops arriving packets randomly with a low probability. The drop probability increases with increasing average queue size and all packets are dropped once the average queue size reaches an upper threshold. The drop policy is based on the *average*, not instantaneous, occupancy thus allowing queues to perform their primary task of absorbing bursts.

The other two authors got involved with RED work while working at the then Bay Architecture Lab. In the course of answering some specific internal questions, we posed the general question "Shouldn't there be

[1] [1] suggests that packets be "marked" by a queue management algorithm. Dropping a packet is the most severe form of "marking" or indicating to an end-system that congestion is present. Although other types of marking may be used [ECN], we have focused here on dropping and will use that terminology.

some simple way to set the parameters based on bandwidth?" and ended up scrutinizing the control law, the sampling, and other aspects of RED and, in the process, starting a collaboration with Jacobson on this, RED-light. Our early results were reported upon publicly in [3] in June, 1998 and in some internal Bay Networks Architecture Lab reports the same year. In 1999, an early draft was made publicly available and more internal reports were published at Cisco. Subsequent work has changed some of our early thinking and there are several areas that could use more exploration. This paper is offered as an aid in understanding how to look at the practical aspects of queue management.

Paul Baran invented much of the foundation of packet switching while at RAND in the early 1960's. (www.rand.org/publications/RM/baran.list.html). In RM-3638-PR (1964), a Communications Control Console (or Priority Control Console) is described. Baran explains how such a device would work in operation:

> "since gross changes in loading are slowly occurring phenomena; rather, he will normally leave the controls set to fixed positions, except when a crisis or overload approaches as indicated by the red warning light. He then decides which users with growing demands should deprive others with less important duties, and to what degree. "

[1] explains the general principles involved and gives simulation studies for a few traffic scenarios. However, in 1993 the Web (http) traffic didn't exist and a 1.5 Mb/s T1 was considered a high-speed WAN link. Since none of the RED parameters were explained in terms of measurable network or transport properties (e.g., link bandwidth or round trip time), it has remained far from clear how to configure RED appropriately for today's wide range of bandwidth and traffic types. Tuning RED for any particular link remains a black art.

The literature on RED that has accumulated since 1993 has looked at performance of RED or alternatives to RED under particular loads. We were driven by the desire to come up with a generalized RED with parameters pegged to pipesize. It's possible to tune our results for one kind of load or another, but they give the best performance across a range of traffic loads.

This paper is an attempt to dispel some of the darkness. All the RED parameters are derived in terms of fundamental, measurable, network properties, resulting in an entirely self-configuring RED. The parameter derivations are validated via simulation and their sensitivity to mis-configuration and stochastic variation is characterized. Finally, a new reference implementation for the RED algorithm is presented that is substantially simpler and faster than existing published implementations.

The work described in this paper was started to dispel some of the confusion. RED parameters are derived in terms of the link bandwidth. More work could be done but we demonstrate that this RED algorithm is much more stable across traffic loads and is much more of a "controller".

2 Input to the Controller

Figure 4 shows the network regulator monitoring queue size. Our input signal is a sample of the queue size and the smoother needs to filter this signal so that the controller doesn't respond too quickly or too slowly to changes in the controlled value (queue size). There are a range of choices available for the queue sample and for the smoothing algorithm. These are covered in this section.

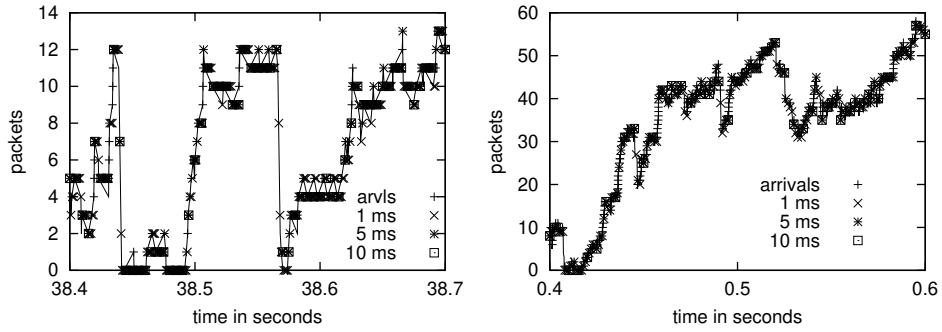
2.1 Choosing an input signal

In [1], queue samples were taken with each arriving packet and an interpolation was used after idle periods. Taking queue samples at packet arrivals has the advantage of "speeding up" the filter if arrivals come quickly, but it also decouples the filter action from time and is not easily implemented on high-speed routers. When the filter input is the queue value at each packet arrival, the filtered value will be recomputed at each packet arrival, so that each arriving packet in a burst will result in a new filter computation. When the filter input is time-based samples, a large burst that arrives within one sample time will be used as one data point into the filter. A time-based sampler does not need to interpolate values when there are no packet arrivals (though there may be departures). The RED implementation shipped with *ns* [] interpolates zero queue values if a packet arrival finds an empty queue. This could introduce significant inaccuracy since the queue may not have been empty all that time and because no interpolation is done unless the queue empties.

Although responding quickly to a large number of packet arrivals could be a useful characteristic of a controller, the disadvantages outweigh any advantage. These include that fact that a large burst of packets

may be dissipated fairly quickly if there are no other arrivals or if packet sizes are small, but the filtered value might still result in dropping the next arriving packet unless the queue has completely emptied.

Packet arrival sampling means an update with the addition of each packet. As a starting point, we consider sampling at periods close to the time it takes to transmit a packet on the link. The sampling itself introduces some smoothing or filtering of the queue occupancy. Look at packet arrivals and sampling for two mixed traffic traces, one at T1, one at 10 Mbps. The RTT averages 100 ms.



Since we are sampling the queue size, the samples can be taken less frequently than the MTU-time of the output link, a particularly useful attribute for high-speed links. As aliasing can result when the sampling intervals are fixed, we recommend that an average sampling interval be picked and a suitable random algorithm be picked. For more discussion of random sampling see the discussion in [9]. In our simulations didn't see value of this, so didn't take up simulation time on it, but should be looked for in "real life." In practice, sampling at several MTU-times is a bad idea for bandwidths less than 10 Mbps. Probably better to sample at less than an MTU time. Our guide would be 1 ms samples. Used 4 ms for T1s. Probably suggest using something on the order of an "average packet time" for bandwidths less than 10 Mbps.

2.2 Smoothing the input signal

Next we consider how best to smooth this input signal. This filtered value is the input to the control law and must indicate a building or draining queue so that the controller can take action, but arrival bursts that can be cleared by the output link without resulting in a queue of packets that persists ought to be smoothed. For example, if the traffic load is a well-behaved TCP in steady state (after slow-start), the filtered signal should show the single packet per round trip time increase in the queue size. That is, the smoother should ignore all bursts that the queue can clear in an RTT while finding the queue length that has persisted over the RTT. Then it is easy to see that *for a TCP in steady state* a filter could smooth over a round-trip time.

The majority of TCP connections in the Internet never reach TCP steady state and thus are more volatile (during slow-start). This suggests a smoother must capture shorter-term variations than a round-trip time and that it must be somewhat robust to variations in RTT. Further, since traffic is generally *more* volatile than a TCP in steady state, permitting a burst of a pipesize every RTT could result in carrying rather large persistent queues. In addition, the round-trip time for connections through a link is generally unknown and varies between individual connections sharing the same bottleneck link. Further, traffic mixes are much more arbitrary than multiplexing small numbers of infinite FTPs and are not known apriori. Although one approach to controlling queues might be to infer traffic mix from the filtered monitor signal, this is a very complex approach and recall that what we really need to do with our control law is to find an appropriate operating point based on the monitored signal.

By persistent queue, we mean that level of packets that does not get cleared in the time period of interest rather than the arithmetic mean of the bursts over that time. To make this clear, consider a queue to which a burst of packets arrives once every T, but which completely clears from the queue just before the arrival of the next burst. The mean value of the queue occupancy is half the burst size, but the persistent queue is zero. Similarly, if the queue size never completely empties, but drops to a minimum of a single packet, we'd like the filtered value to reflect that. If the queue size were only increasing, then the average value over the past interval T might have some value, but it misses the draining queue.

In [1], the smoother employed is an exponential weighted moving average (EWMA), or a low-pass filter (LPF), allowing for efficient implementations as in [8]. [1] uses a filter gain of 0.002 (called *queue weight* in [1]).The EWMA equation is:

$$F_k = (1 - g) * F_{k-1} + g * Q_k$$

where g is the gain of the filter, less than or equal to 1, F_k is the filtered queue size at sample time Q_k, and is the sampled queue size at sample time k. Making no assumption on arrival bandwidth (i.e., packets can arrive arbitrarily fast), we explore filter behavior analytically. A simple case is when there is some constant queue size that persists; for example, the queue reaches a level of C packets and thereafter for every departure there is an arrival to take its place. Using the unit step response of the filter, the filtered value after n samples have been taken is:

$$F_k = 1 - (1 - g)^n$$

which we can solve for the 90% value N, that is the number of samples to get reach 90% of the persistent queue as represented by the step input.

$$N = \frac{\ln 0.1}{\ln(1 - g)} = -\frac{2.3}{\ln(1 - g)}$$

Dividing n by S gives the number of round-trip times. Sampling the queue at intervals equal to the time to send an MTU-sized packet at the link bandwidth, implies that $S = P$. Setting g to 1/P, the 90% response level is reached in a little over 2 round trip times. If the actual round-trip time is twice as long as the assumed value, then it will take about a round-trip time to respond. Setting g larger permits a closer tracking of the building queue, but at the risk of following transients we'd like to ignore. Consider a T1 link on a path with a 100 ms round-trip time and thus P = 13, compare N for a gain of 0.0625, a value very close to 1/P, to the gain value of 0.002 recommended in [1]. A gain of 0.0625 yields 36 samples or 2.7 round trip times to reach the 90% level and a gain of 0.002 yields N = 1,150. For a 1.5 Mbps link and 1500 byte packets, that's over 90 round trip times; for 512 byte packets, more than 30 round trip times, a filter that responds too slowly.

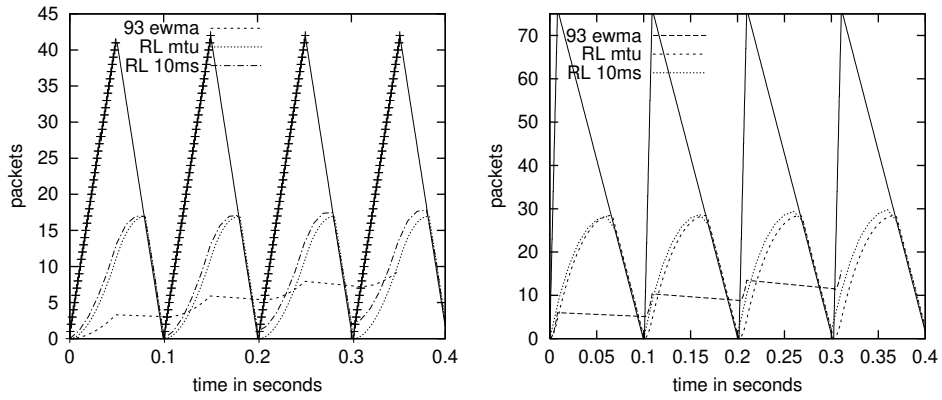
The filter must also handle the downward fluctuations in queue size properly and track the persistent queue. A queue that is empty at least once every round trip time should result in a filtered queue size of zero. The lagging properties of the LPF make this impossible, thus a different filter is required when the monitored signal shows that queue size is decreasing. One way to accomplish this and to respond more quickly to a draining queue is to set the filtered value to the queue sample when the latter is smaller than the former. This is equivalent to changing the filter gain to 1.0 when $Q_k < F_{k-1}$. Then we can write:

$$F_k = F_{k-1} + g * (Q_k - F_{k-1}); Q_k > F_{k-1}$$

$$F_k = Q_k; Q_k \leq F_{k-1}$$

If we chose the gain to be the inverse of the number of queue samples in a round-trip time, the filter will average over a round-trip time and approximate the mean over the round-trip time. If the average sample interval is the transmission time of an MTU, this gain should be the inverse of the pipesize in MTU-sized packets, 1/P. More generally, if we sample the queue S times in a round-trip time, our filter gain should be on the order of 1/S. So we chose RTT/sampling_interval rounded to next higher power of 2, but 2x for bw of 10M and up and 4x otherwise. Need fast response. Believe real answer is a better filter.

First, look at artificial pattern. A burst of 83 packets arrives each 100 ms. The departure rate is 1.2 ms (an MTU time at 10 Mbps) and the arrival rate is twice as fast, 0.6ms per packet. Use a 93 EWMA (with interpolation after idle times). Its interpolation acts as though there were that number of zeros. The RL filter uses a gain of 1/S (rounded to next lower power of 2) or 0.015625. For comparison, we show sample intervals of 1.2 ms and 10 ms.

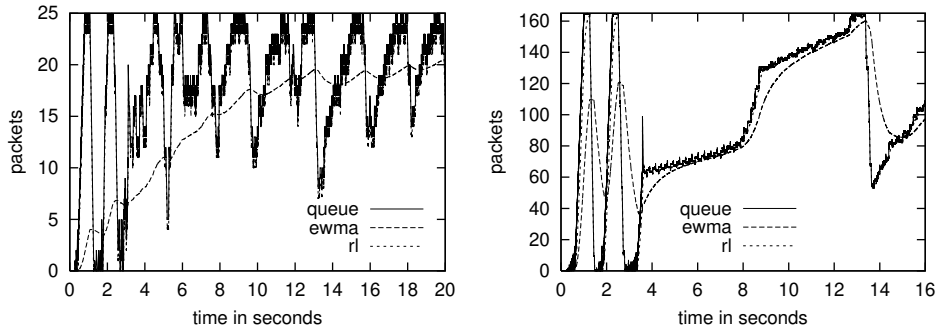


The RL filters track the queue better, though can see that minor variations happen with differences in where arrivals and departures fall relative to sample times. The 93 filter has a slow gain which is evident here. However, the overall trend is upward. It is approaching the average queue, or a value of 21 packets. A smaller gain will make this happen faster. Next we change the arrival rate to 0.1 ms.

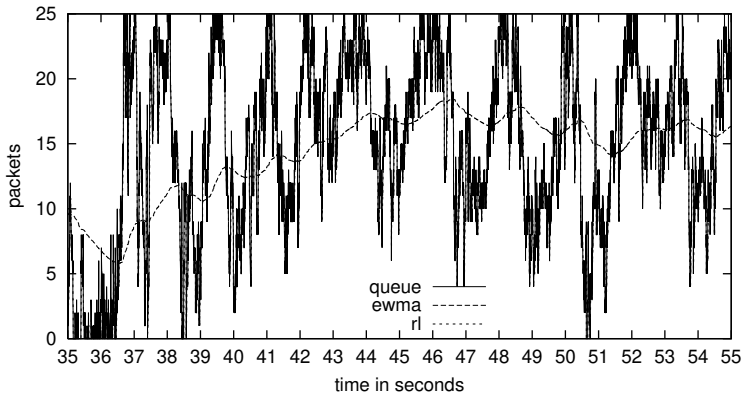
Note that the EWMA keeps increasing. Reason is that the time constant is so long due to the small gain that even 80 averaged in idles ((1-g)**80) is about 0.85, so that the peak value of 6.0 goes to 5.1 and then ramps up. The second shows different ways of sampling. Also tried sampling at 10 ms intervals with little difference.

2.3 Filters in action

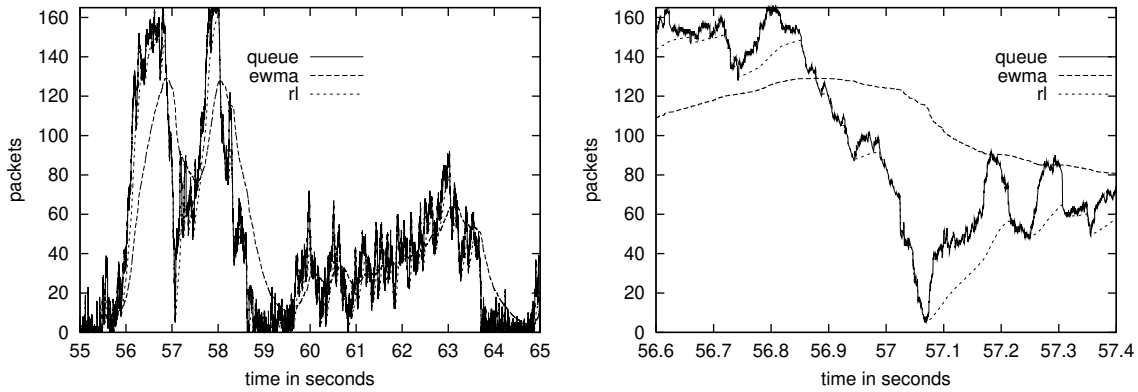
Next we show filters in action on portions of some simulation traffic traces through drop tail queues. Recall that when the instantaneous queue is determined from packet arrivals alone it will differ some from the packet queue computed at sample times. The former will tend to miss departures and thus may be a bit higher though they are very close for these examples. For uniformity, we use the packet arrival inupt for both. The figure shows packet queue, the 93 ewma filtering, and the RL (red light) filter. Figure a shows a 1.5 Mbps link with three "infinite length" TCPs through it. The start up phase of the three is included and a 20 second period (200 round trip times) is shown. RL tracks the queue quite closely, nearly indistinguishable at this time scale. Though there may be a temptation to chose the very smooth looking 93 filter, note that it would indicate a large queue for packet arriving to a relatively small queue during several time periods, for example just before 51 seconds. For the 10 Mbps bottleneck of figure b, the gain of 0.002 tracks its five "infinite" TCPs somewhat closer. Still the lag is quite apparent.



Next consider a varied traffic pattern (mix of ftps and webs) through a 1.5 Mbps link:



Notice there are several seconds where the ewma records a value with a high drop rate when the actual queue is low. For varied mix through a 10 Mbps link, we show a 10 second period as well as zooming into a one second (or 10 round-trip time) period.



Looks smooth, but note several *seconds* of lag.

3 Putting it to work: results

In the previous sections, we explained how a regulator for internet traffic should work and developed a procedure. The difficulty in implementation is covering the wide range of adaptive and non-adaptive traffic of the internet. We simulated a range of operating conditions and traffic mixes in order to gain an understanding of RED and parameter setting. RED-light (RL) parameters are set as discussed in the previous section. Our aim has been to explore the tradeoffs with changing parameters and the robustness of our algorithm.

We show performance against the RED control law proposed in 1993 (93) and parameters recommended for RED in [1] and notes [7] (93g) from March, 2000. We set $\max_p$, the value that determines the slope of the drop curve (93's control law), at 0.1 as recommended in [7] and the 93 "gentle" curve continues linearly to a probability of 1.0 as the average queue gets close to the maximum buffer size. [2] Though the 93 RED parameters were originally not tied to pipesize in anyway, the 2000 notes suggest a min_{th}setting of $buffer\ size/12$and a $max_{th}$of $3 * min_{th}$. We've used those parameters and the *ns-2* 93 RED. The values of $min_{th}$and $max_{th}$are used as lower and upper bounds to persistent queue values, so our results explore their efficacy.

Our work with RED has stretched over a number of years and in that time we've employed a number of metrics for performance and design of queue management. Results presented here focus largely on how

[2]In 1997-8, Poduri and Nichols experimented with changes in the 93 RED control curve, including a "two-part" curve, but this exhibited tuning problems at both ends. This work was done in the Bay Architecture Lab and written up only in internal technical reports. 93g is related to this and has the same problems.

well the queue is controlled. Performance of each controller on queue size management is indicated with the median queue size and the 90th percentile queue size for each run (shown in milliseconds or bytes in queue times bandwidth of bottleneck link). To see how well the the queue management is tracking the queue, the distribution of all the queues sizes over a run and the distribution of the queue sizes at drops. The queue size samples are updated for each packet arrival or departure and idle queues are interpolated at MTU-times. Good queue management should not drop arriving packets when the queue is empty or nearly so. For the RL QMs, there should be few drops below the lower bound.

The simulated network is not quite a "dancehall dumbell"[] with clients on one side and servers on the other. There are reverse-path clients and servers which are included in the traffic mix, but not in the results presented here. RTTs are varied around 100 ms (from 75-125 ms). The TCP implementation used an "ack every other packet" policy to mirror the most common TCP implementations. We look at three bandwidths: T1 (1.5 Mbps), Ethernet (10 Mbps), and T3 (45 Mbps) and used the recommendations above to set parameters. Buffer sizes are set to $2 * bw * d$ for links of 10 Mbps and smaller and to only a single $bw * d$ above that.[3] The table gives the exact values used (The RLB values are converted to MTU-sized packets for comparison).

B	93	RLP	RLB	B	93	RLP	RLB
1.5 Mbps	2.2	2	2	1.5 Mbps	6.5	7.8	6.5
10 Mbps	13.8	8	8	10 Mbps	41.5	42	29
45 Mbps	31.3	30	15	45 Mbps	93.8	131	94

3.1 Queue regulation for FTPs

The original paper on RED used a traffic load of TCP connections in steady-state, a load that might be created by long-lived FTPs. This traffic load is easy to analyze and a controller for a buffer that only carries such a traffic load is easy to design. Unfortunately, though most of the traffic of the Internet uses the TCP protocol, only about 10% of flows are at all long-lived (that is, even make it into TCP steady state). Still this mythical traffic load, known as *elephants*, has been widely used to test and justify a range of possible controllers. Though TCPs account for 60-90% of the *byte*s seen in backbone links, controlling elephants is not sufficient as we shall see. Further, the links most likely to be bottlenecked in today's Internet are more likely to have a higher share of short, HTTP style transactions.

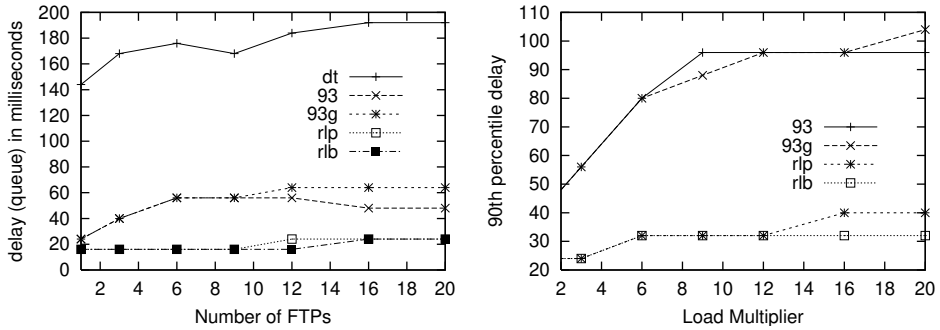
The elephant traffic load can be used as a way of understanding controller performance under ideal conditions. In addition, we add another FTP-style traffic load which we term *moose* to exercise controller tracking. Moose are FTPs that transfer a smaller file, thus each client goes through slow-start, steady state, and closing of several TCP connections during a simulation. It is much more difficult to track the slow-start phase than the steady state phase.

3.1.1 Elephant herds

As we have seen, each drop has a lot of leverage for a single FTP. As more connections share a link, the window size any connection can have open decreases until the connections are being regulated by the timeouts, rather than TCP's congestion avoidance. Thus we move from high to medium to low leverage per drop as we add FTPs to the traffic mix. Simulations start with a single FTP through the link and runs of increased numbers of FTPs are run till the link is quite saturated

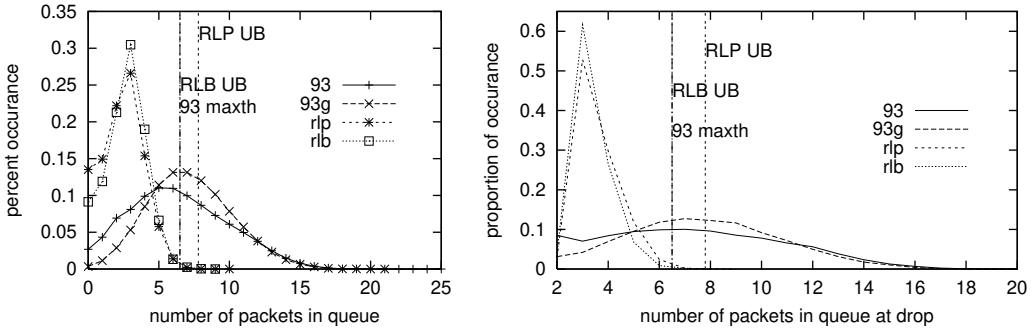
1.5 Mpbs link. The bandwidth delay product is 12.5 for this link, so after 12 clients, there are more FTPs than there is space for packets in the pipe. The figures show both median queue and the 90th percentile queue in milliseconds vs the number of FTP clients. For the median, the drop tail (dt) queue results are shown for interest, but not for the 90th percentile.

[3]At bandwidths of 100 Mbps and above, buffer sizes are frequently reduced further.



Note that the 90th percentile delays for the RLs is better than the median of the 93 QMs. Since a major goal of QM is to control delay, the 93s are less suitable for this. Further, there is an "overload effect" for the 93 after the 12 client mark. What happens is that the majority of the drop are forced drops, above the control region.

Since the controller for the 93s and for RLP looks only at the number of packets in the queue, the quality of the controllers can be assessed to some degree by looking at the density of the sampled queue sizes. The next figures are all the samples for all the loads for each QM. The upper bounds are indicated on the graphs to see how well the controller kept the queue size in range. (It is a filtered version of the queue history that is compared to the bound, not the instantaneous queue.) To compare RLB, whose controller is not based on packet size, the size of the RLB queue in bytes is converted to MTU sized packets. The second graph shows the number of packets in the queue when drops occur. No drops occur when there are fewer than 2 packets in the queue for any QM, with about 8% of the 93 drops at 2 packets and about 3% of the other three QMs drops at 2 packets. The 93s don't do a very good job of keeping the queue below max_{th}. For 93, the packet queue samples exceed max_{th}36% of the time; for 93g it is 44% of the time. For the RLs, the number of samples above the UB is negligible, 0.003% for RLP and 0.01% for RLB. The maximum queue sample recorded is 25 packets for 93, 21 packets for 93g, 10 packets for RLP and 9 packets for RLB.

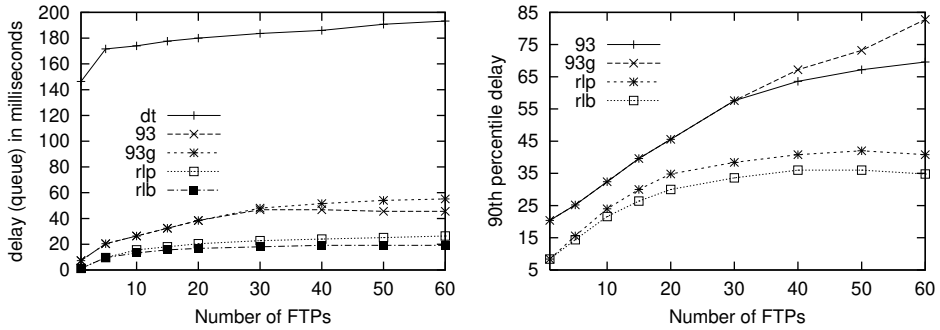


The distribution of the queue and failure of the controller can also be noted by the proportion of packet drops that are *forced*, that is outside the controlled region. The table shows the median across all the loads and the range. There are no forced drops for the rls.

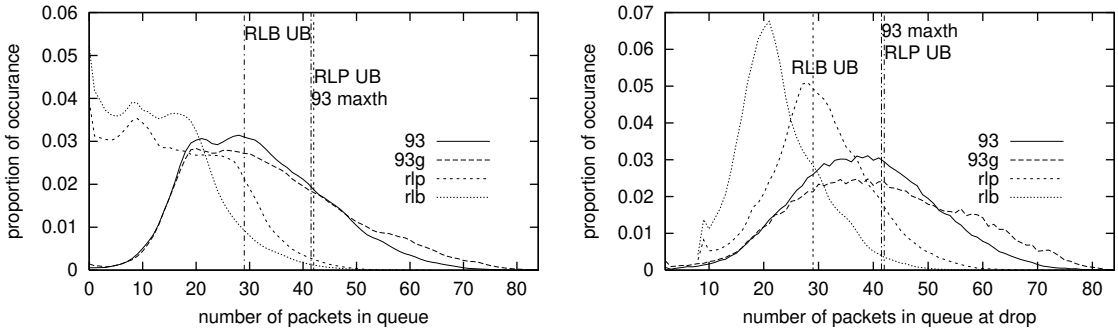
QM	largest pq	%pq above max	med forced drops	range of fds
93	25	36	61%	2-80%
93g	21	44	0.9%	0.6-11%
rlp	10	0.003	-	-
rlb	9	0.01	-	-

A 10 Mbps bottleneck

The larger pipesize gives a larger control range.



The packet queue is more frequently below the maximum for the 93s, but a large percentage of the drops occur above max_{th}.

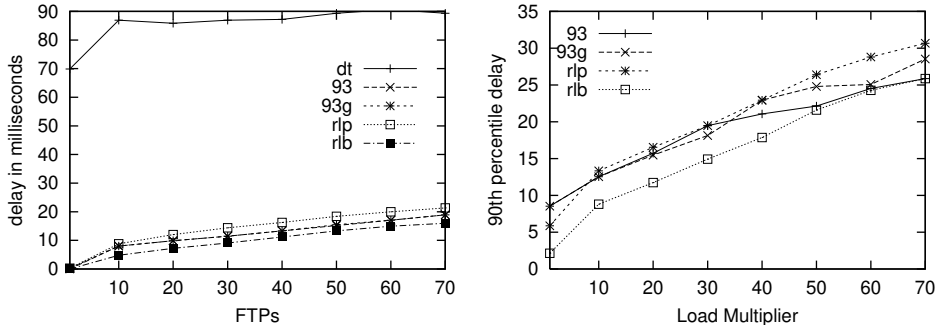


The table shows the largest packet queue, percentage of samples above the maximum or upper bound and the percentage of drops made when the packet queue was below the minimum. A value of "0" indicates a very small result, "-" indicates none found. For 93 more than 41% of samples were above the maximum for the largest three loads and for 93g, more than 50% of samples were above maximum for the largest three loads. Further, it is possible for an arriving packet to experience a delay considerably beyond the configured maximum. This is not the case for the RL controllers.

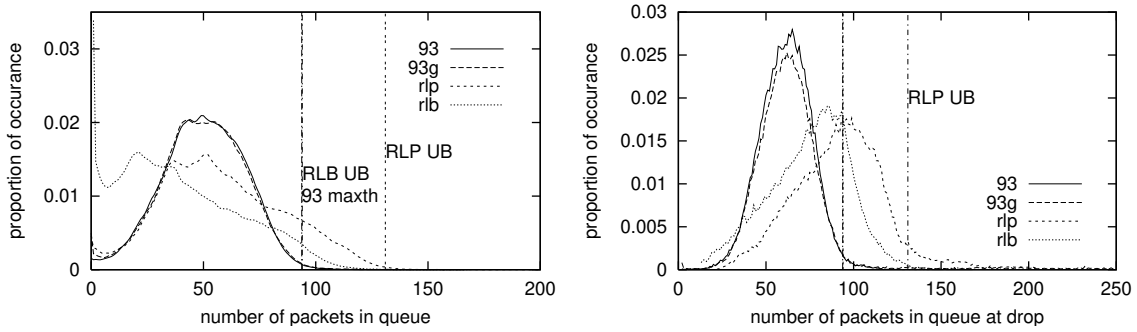
QM	largest packet queue	%above max	%drops below min	range of fds
93	144	19	1.6	3-64%
93g	166	25	2.4	0-17%
rlp	64	1	-	-
rlb	62	0	-	-

A 45 Mbps bottleneck

The reduction of the buffer size to a single bandwidth delay product affects the min_{th} and max_{th} of the 93s. In turn, this leads to smaller delays, but more forced drops, that is drops that happen beyond the control range.



The packet queue distribution for the 93s is much improved with samples above max_{th}only 0.4% of the time for 93 and 0.8% for 93g. RLP exceeds its upper bound 0.2% of the time and RLB 3% of the time. Although RLB exceeds its bound the most often, these values are not large as can be see from the maximum packet queue sizes of 166 for 93, 375 for 93g (vs 205 for RLP and 168 for RLB). The 93s do much more dropping with very small queues, with 5% of 93's drops when there is a queue size from just above 0 to 5 ms and 3% of 93g's drops in the same range. For RLP, there are a negligible number of drops in the range from 1 to 6 ms (none below) and for RLB, a negligible number of drops in the 3 to 6 ms range.



QM	largest pq	%above max	drops below min	range fds
93	166	0.4%	1.9%	4-98%
93g	375	0.8%	1.8%	0-42%
rlp	205	0.2%	0.3%	-
rlb	168	3%	0.2%	-

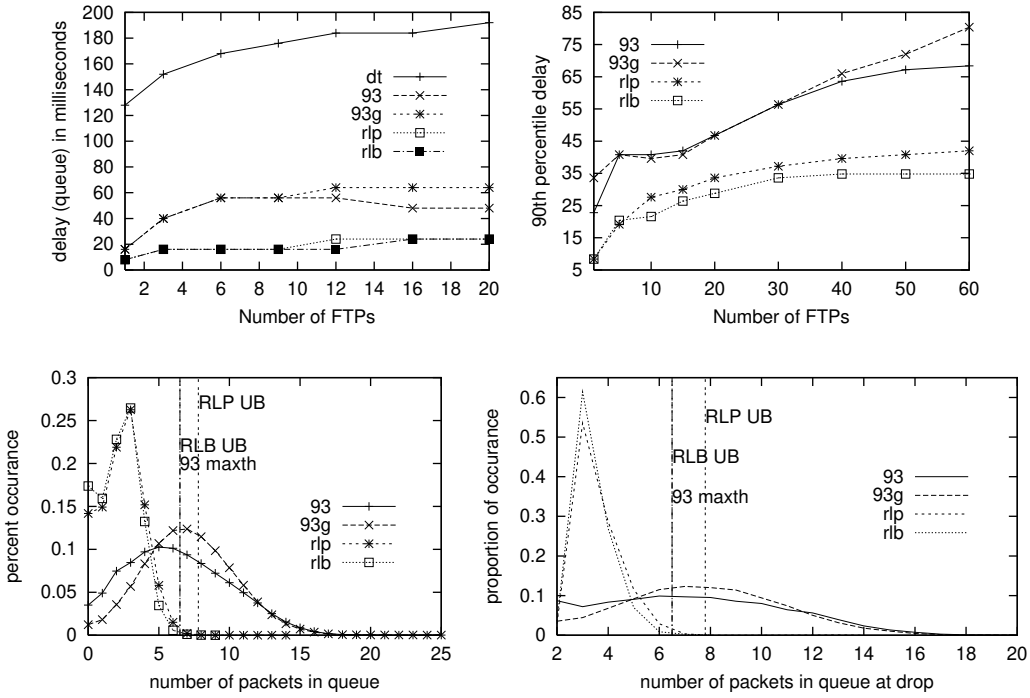
Fairness, forced drops and other interesting stuff

The topic of "fairness" is sometimes brought up with RED. This can only be applied meaningfully where the traffic load is rather uniform, as for elephants. It is quite tedious to look at such measures and we have found reasonable fairness across all schemes, including DT (though sometimes an outlier or two). Back to controller stuff, we'd like our controller to actually come into play most of the time (unless there is negligible queue). So, "forced drops" which indicate those that occur outside of the control range are to be avoided. RLs do an excellent job of this and 93g is better than 93 due to its increased range. The 93 QM is frequently out of the control range (this also indicates that the 93g is dropping on the steep part of its control law curve an equivalent amount of time).

insert table of values?

3.1.2 Moose in the fields

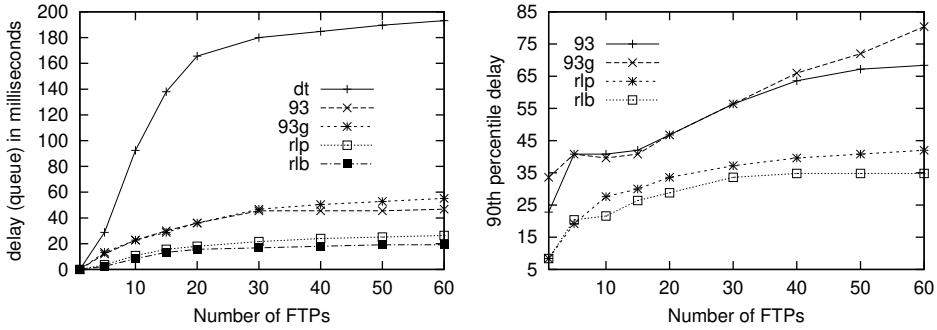
Long-lived FTPs are very predictable and such herds of elephants (very long time in steady state compared to observation time of interest) rarely occur in real networks. Moose are FTPs that have time to get through slow start to steady state and to close in a fraction of the simulation time. Shorter transfers means more start up phases. It is also unusual for network traffic to consist soley of packs of moose, but the varying traffic patterns puts more realistic stress on queue management schemes. For T1 median delays, the median delays mirror those of the elephants, but the variability in queue delays has grown.

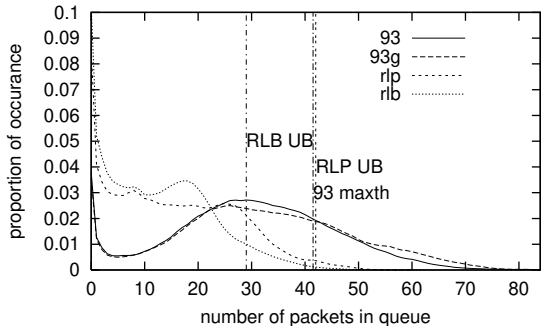


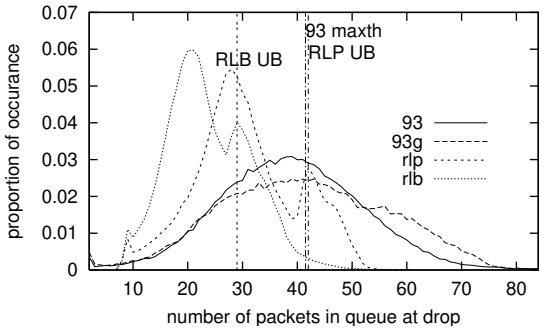
The results with packet queue occupancies are also similar, with the "upper bound" for the 93s being a maximum only in a very loose sense. Although 95% of the time the queues are less than a pipesize of packets, there are times they grow to the maximum, 26 packets. This is not a good situation since the controlled queue should remain below the pipesize. The RLP queue does grow to 14, but is below a pipesize 99.99% of the time and the RLB never has more than 9 packets in its queue. 93 is usually doing forced drops: only one load is at 6%, one at 34%, the rest are over 50%. 93g only has one load (single FTP) at 66% forced drops, one load at 5% and the rest are 1%.

QM	largest pq	%q above max	range of fds	
93	26	36	6-79	
93g	26	44	1-66	
rlp	14	-	-	
rlb	9	0.1	-	

Next we look at 10 Mbps bottlenecks:







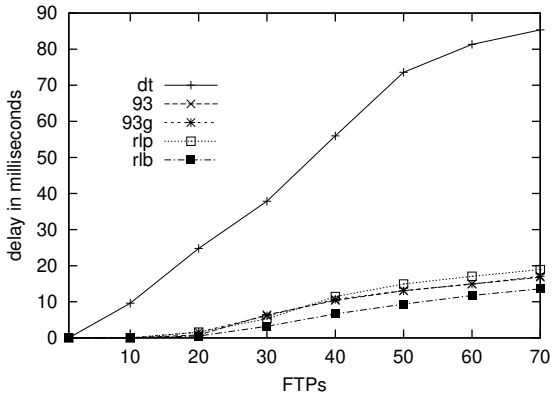
In the single FTPs, although the two RED lights had roughly twice the drops, both got 99% utilization for the single FTPs and the 93REDs got 97%. This is due to the drops in RED light being more effective. In addition, the queue variation was smaller for RED light.

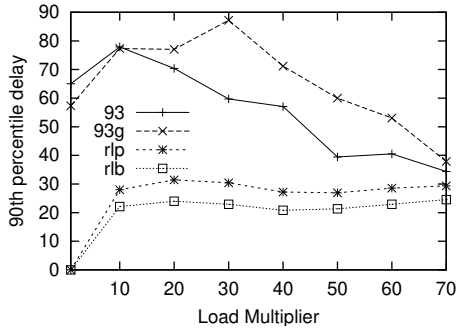
size of q at time of drop cdf

QM	largest pq	%drops below min	range of fds
93	166	2.4	13-63
93g	166	2.8	0-19
rlp	60	0	-
rlb	62	0	-

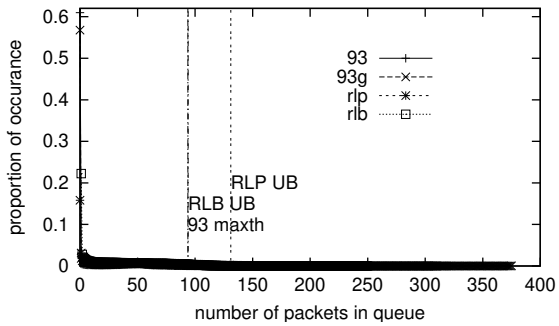
Here there were a small number of RL drops when there were 7 packets in the queue (one below the threshold and within the canceling range), the 93s had drops for queues as small as 2 packets.

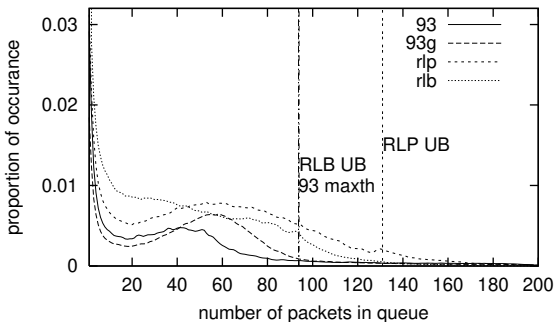
For T3

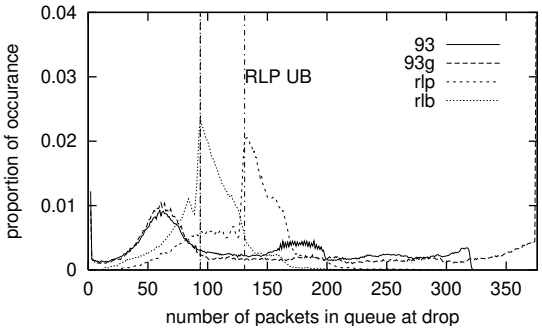




Looking at the distribution of the queue, find that 61% of the sampled time the 93g was a zero queue and 59% of the time the 93 was empty, 16% of the time for RLP (never for RLB, first point is 22 of the time queue size of 1 packet). The largest queues seen for 93g was 375, for 93 was 322 for RLP was 310 and for RLB was 250. Second plot is "zoomed in"; left off the zero points and outliers.







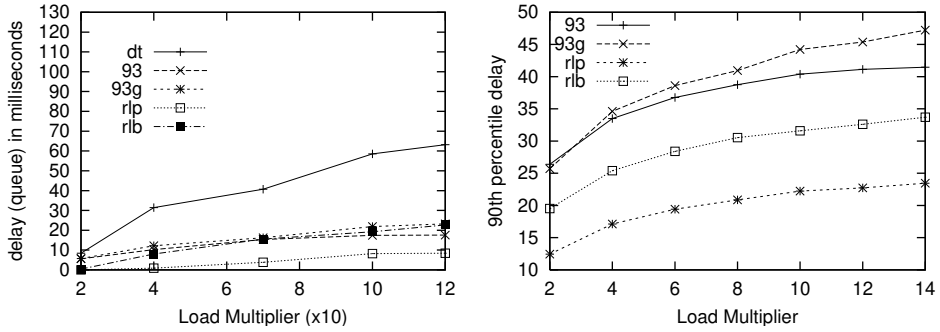
QM	largest pq	%q above max	%drops below min	range of fds
93	322	5	2.4	24-93
93g	375	4	2.8	0-60
rlp	310	4	0	-
rlb	250	7	0	-

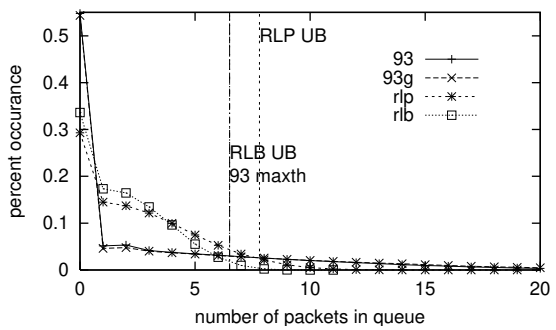
3.2 Web mice underfoot

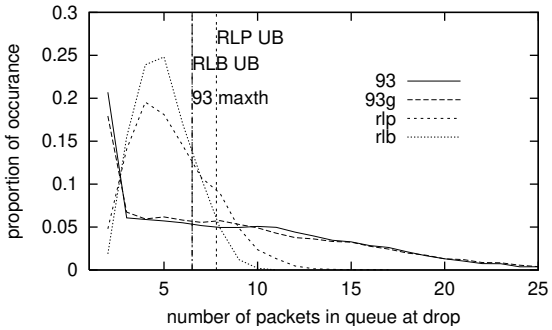
If the congestion is due to web traffic, a SYN drop has the highest leverage (gives the highest medium term bandwidth reduction) since it will postpone all traffic on the connection for 6 seconds. Dropping any other packet will only delay a portion of the traffic for 100-500ms. An ack drop has the lowest leverage (since acks are cumulative) but ack-dropping is a low probability event since paths rarely get congested with acks (the forward or data path almost always congests first) and, in mixed ack and data traffic, the ack packet density tends to be at most half the data packet density (since the most common receiver ack policy is at most ack every other packet).

In some ways this kind of HTTP 1.0 represents a "worst case" for queue management in that the queue sizes can change quickly. On the other hand, it is long-lived TCPs that push up a buffer size, as can be seen clearly from drop-tail results for ftps, above.

Results of experiments with multiple web connections though a T1. The results show that it's much more difficult to control the persistent queue for bursty traffic like web accesses although RED light does still manage to run without full buffers. Further we note that the web model we use is quite pessimistic in a network traffic sense. We use a model that has three simultaneous TCP connections opened for "in- line" transfers. Queues are controlled better if this is not permitted or only represents part of the traffic mix.





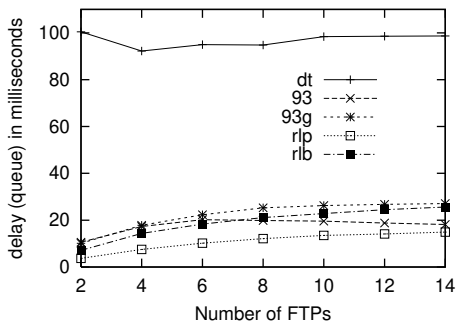


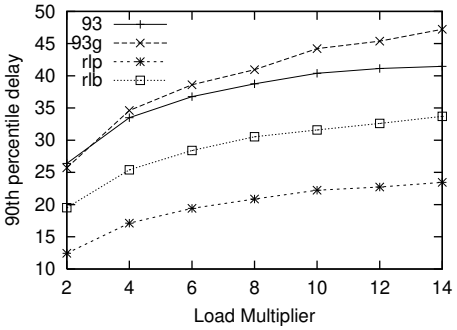
QM	largest pq	%pq above max	range of fds
93	26	17	14-72
93g	26	19	3-7
rlp	19	2	-
rlb	11	3	-

There were no drops with fewer than 2 packets in the queue.

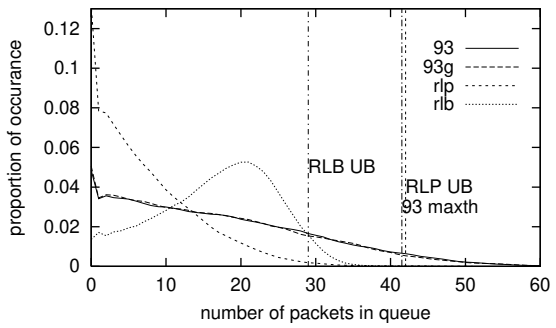
Web connections through a 10 Mbps link congest the link for number of clients at 150 and up. Although we repeated these experiments with the three sampling schemes (random sampling with a 7 ms average, fixed sampling at 7 ms intervals and fixed sampling at MTU or 1.2 ms intervals), there was no significant difference between the results. We report only the MTU sampled results. Again, the web traffic is more demanding of our regulator than the FTPs, but roughly half the queue is available for traffic bursts even at congested levels and note that the rate of increase of the median slows. We use a web-browsing client model that opens three simultaneous connections after the response is received from the initial (or primary) URL ([11] has more discussion of this model). This makes for a quite bursty traffic pattern which is greatly mitigated by going to only two simultaneous in-line connections.

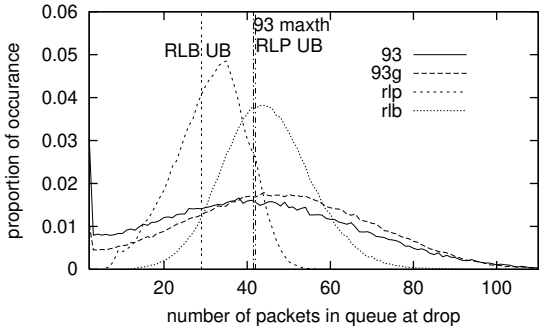
10 Mbps medians





Also the pq cdf(truncating dt) and the delay cdf





range

QM	largest pq	%pq above max	range of fds
93	75	4	2-78
93g	82	4	0-1
rlp	51	-	-
rlb	46	0.6	-

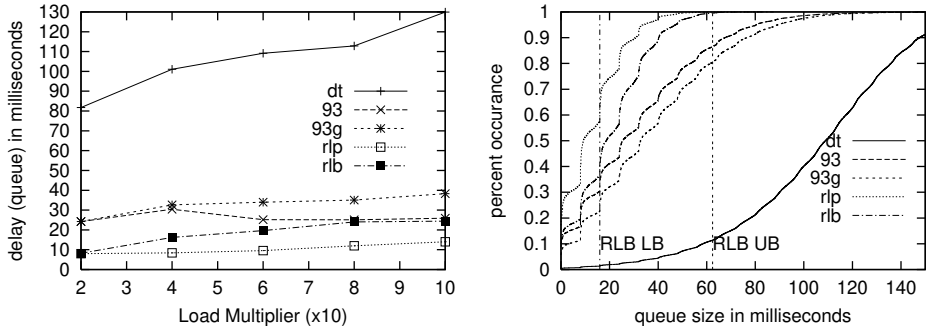
The 93 has a lot of forced drops, the largest 4 loads being above 60% forced, and the load value of 6 having 44% forced drops. Most of the 93g runs have no forced drops.

How does 93 RED do? First look at the results where $\max_p = 0.1$, as recommended in [7]. For uncongested or less congested cases, the control is similar to that of RED light, though the drops happen, in general, at a slighly lower queue size in 93 RED than for RED light. Once the link gets congested, though, the 93 RED has a more difficult time regulating the link: the median queue size grows toward the top of the controlled range. Many of the drops that occur are at the top of the controlled range.

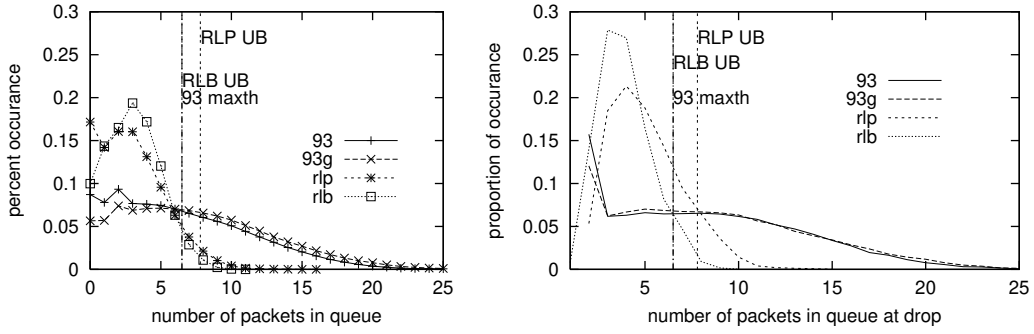
3.3 Moose and mice dance

In this section, we employ a traffic load created by a mixture of HTTP clients and FTP clients. Since the results of sections 5.1 and 5.2 give us a good picture of what happens in extreme overloads, we explore a more intermediate region. For each bandwidth, we used the number of web clients that yielded about 50% and about 85% utilization in section 5.2 and added first one FTP, then two FTPs. We looked at several measures of overall performance including page completions, rough fairness, bytes transferred, median queue size and report the most salient here.

Median q



For T1, packet queues:

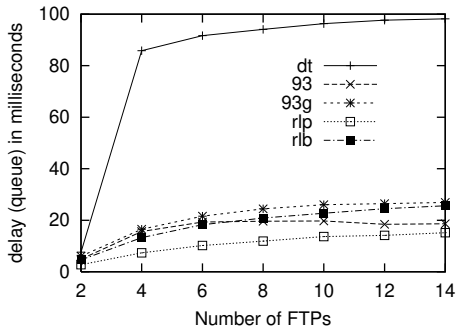


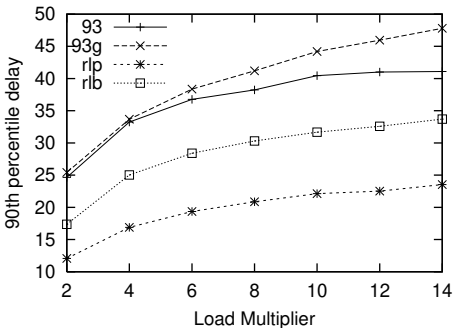
For stampedes as for mice, good to look at time, not just packet queue.

QM	largest pq	%pq above max	%drops at pq=2	range of fds
93	26	33	16	35-78%
93g	26	46	12	1-18%
rlp	16	1.7	5	-
rlb	11	3	2	-

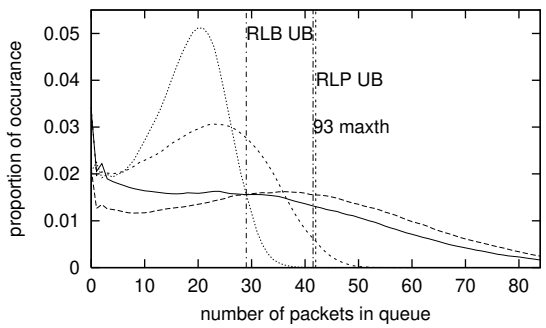
For 93, all runs except 2 had a majority of drops as forced. For q above max, runs 4-10 were above the max 46% of the time or more. For 93g, runs 4-10 were above the max more than half the time. For drop tail queues, the FTPs tend to push out the Web transactions. This can be seen by computing the difference between the share of drops received by web traffic and the share of packets sent by web traffic. For DT, this is always a positive value, ranging from 6% at load 2 to 13% at load 10. For all the REDs, this is bounded by the absolute value of 3% (more often slightly negative for the rls and 93). (RLP has the least variation from zero.)

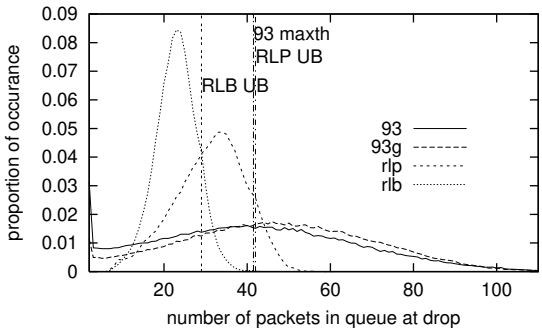
We found that all schemes were relatively fair between the two FTP clients. Web traffic got a nearly identical overall share of the link in both RED-managed queues. In general, web packets got a slightly lesser share of the link for the droptail queue with a single FTP while drop tail with two FTPs clearly gave the webs a lesser share of the link than the RED-managed queues. For example, with drop tail, web packets got 55% of the FTP packet share for ISDN, 75% of the FTP packet share for T1 and E10. This supports our observation that in drop tail queueus, the FTPs fill the queue and HTTP packets often can't fit in the small remaining space and are dropped. This is also supported by examining the HTTP packets' share of the drops divided by the HTTP packets's share of all packet successfully sent. For most of the RED-managed experiments, this is between 1.0-1.1, while in the drop tail queue experiments, the ratio varies from 1.1 to 2.4! Interestingly, for the E10 link, sometimes HTTP sees less than its share of drops.





The packet q pdf



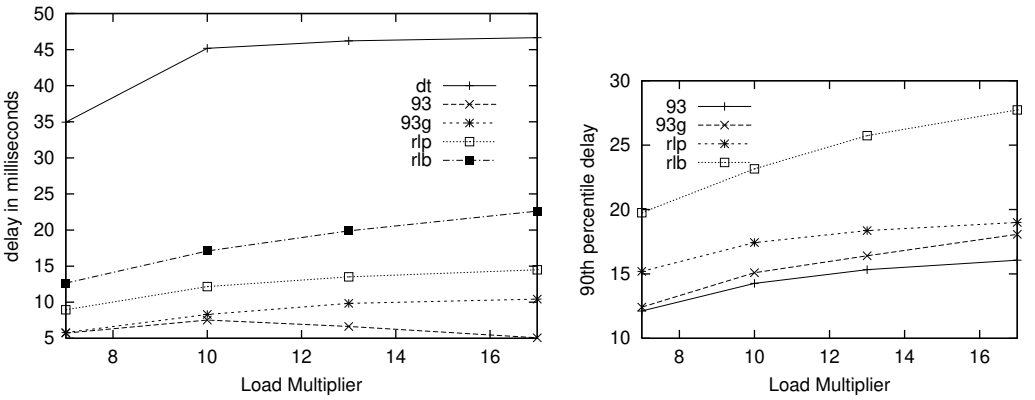


QM	%q above max	largest pq	%drops below min	lrange of fds
93	29%	166	12.5%	23-77%
93g	39%	166	7%	0-10%
RLP	1%	61	0.3%	-
RLB	3%	46	0	-

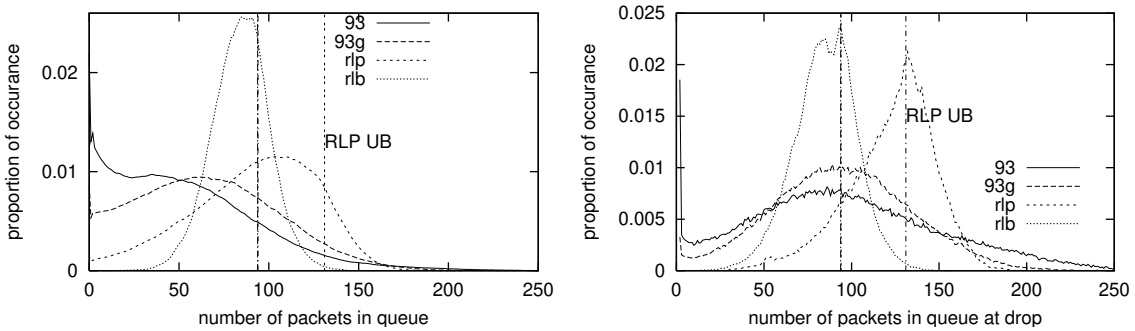
There are no drops by rls below 7 packets in the queue. (7 is the cancel thresh) and the number of drops at 7 is negligible.

The advantages of using RED-managed queues are very clear from this data, but the superiority of RED light over 93RED is also clear. The RED light utilzation is always equal or greater than the 93RED values. In all experiments but one, the RED light drop rate is lower. The total web pages transfered is always greatest for RED light. Although the 93RED median queue is lower than RED light we argue that, in light of the other results, this represents overcontrolling of the queue. Furthermore, the difference never exceeds 12% of the total queue size and is usually less. On the other hand, we note that both RED-managed versions are an improvement over drop tail.

For T3, median delays



Here the 93's perform better on delay since the parameters are tied to bandwidth.



QM	%q above max	largest pq	%drops below min	largest delay
93	16%	281	11%	18-85%
93g	24%	294	5.5%	0
RLP	11%	208	0.02%	-
RLB	28%	159	0.001%	-

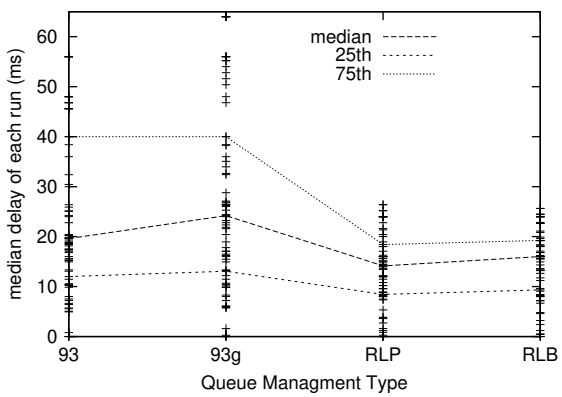
3.4 An overall look at the data

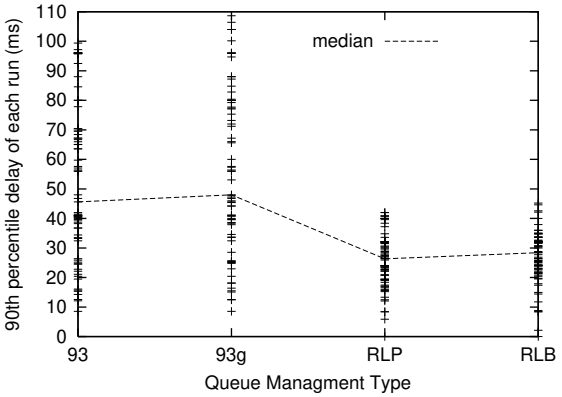
Recall that we are looking for a controller that works well across a range of traffic loads and can be parameterized in a straightforward fashion.

We judge a controller by how well it controls the queue, measured in both time delay introduced and size of the queue in packets. We also look at the cost of the control, here expressed by dividing the delay by utilization (and leaving out the single FTP loads). We also want to look at the size of the queue when the drops occur: does it seem to be controlling the queue or are the drops completely random or responding to "old news"? That is, drops that happen when the queue is nearly empty. We also look at the "forced drops", that is, how often is it getting out of our target control range? Also look at the percent of single-spaced drops, though this is expected to be higher for higher loads.

As we have seen, the cumulative distribution functions of the queue statistics are the most revealing. We can combine all the cdfs for a particular queue managment on one plot to see how much they differ.

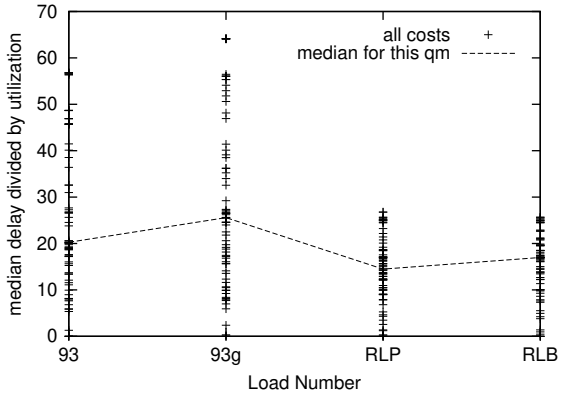
We can also plot all the medians for each scheme and the median values for these.

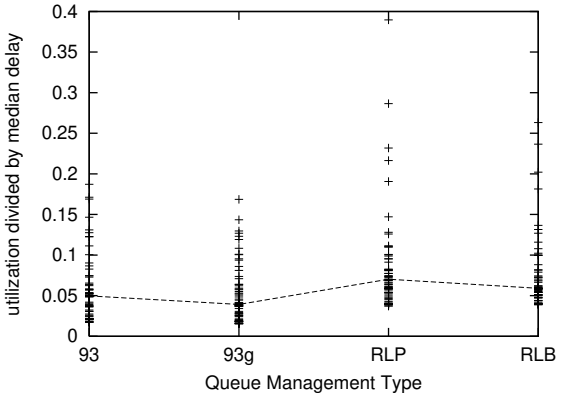




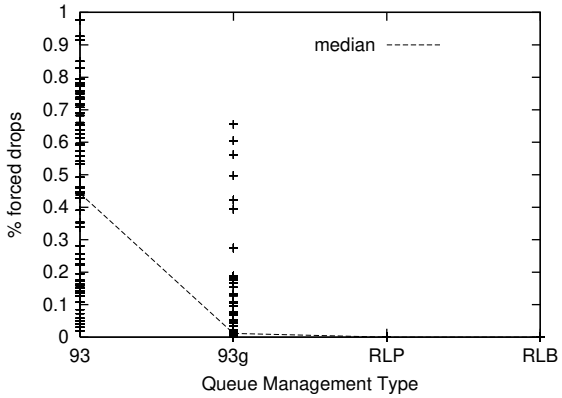
It's clear that the red-lights have a tighter range of medians and that the median of these is lower.

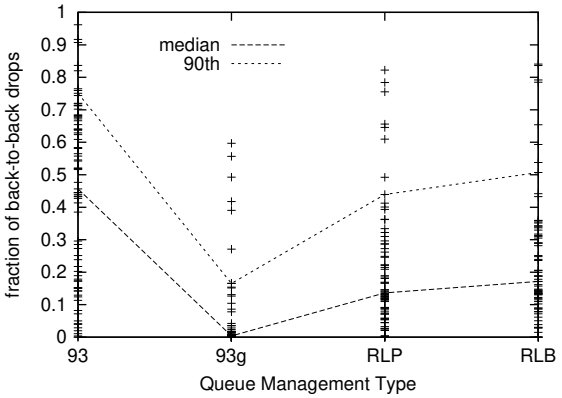
We can also look at a such a simple metric for cost, the related function:





Also can look at the percentage of forced drops:





3.5 When the round-trip delay departs from the parameterized value

In section 3, we introduced some discussion of what happens when the RTT differs from the assumed 100 ms value. We explore this further by setting parameters using the "canonical" 100 ms value and varying the RTTs of the connections using the bottleneck. We first experimented with increasing numbers of FTPs

through a 10 Mbps bottleneck where all connections have the same RTT which we varied from 25 ms to 500 ms. When RTTs are longer than 100 ms, the link utilization at low degrees of multiplexing is decreased. For RTTs up to about 300 ms, the median queue is maintained in roughly the same range as the 100 RTT connections, but deviations are greater. At an RTT of 500, the utilizations are lower and the persistent queue is quite low, though the deviations are large. This can be important for long-delay paths since a large queue adds more delay to an already long delay path. The short RTTs are of more concern since their connections are ramping up more quickly than our "canonical" connection. Does a RED light regulator still control the persistent queue for such connections? Results for the 25 ms and 50 ms round trip times indicate that RED light controls the persistent queue quite well. (The small RTT experiments all run at about 99% link utilization.)

Another issue with varying RTTs is whether the RED light regulator exacerbates the well-known RTT unfairness. That is, for connections of different RTTs sharing a bottleneck, those of smaller RTT generally get a larger share of the link bandwidth. To test this, small numbers of FTPs, each with a different RTT, were multiplexed through a 1.5 Mbps link. We experimented with 8, 5, and 2 flows. The median queue sizes for all experiments remained between 9 and 11 with deviations of 1.5 to 2 packets. We recorded the share of the link each connection got as well as that connection's share of the total drops. The greatest difference was seen when two flows share the link. At slightly higher degrees of multiplexing, the relative shares of the link still differ, but the relationship to RTT is less pronounced. Flows appear to get a share of the drops roughly equal to their share of the link, as desired.

3.6 When the output link bandwidth varies from the parameterized value

Queueing and scheduling schemes may cause variation in bandwidth.

3.7 Summary of results

In the results we saw that the 93RED algorithm could be "tuned" by changing its control range and setting maxp to 1.0 to give "reasonable" performance, but that the performance does not seem to hold up as well over general traffic mixes as our RED light. Further, the setting of parameters appears to be much less robust than for RED light. However, the results and analysis presented in this paper should improve understanding and tuning of 93RED schemes, if it is necessary.

Note that for REDlight it's a simple matter to adjust parameters for "more queue" (that is, more delay) and higher utilization by increasing the threshold. The upper bound makes much less difference. In general, it's difficult to get "perfect" parameters for T1 rates. At T3 and above, the threshold should be dropped if shorter queues are desired.

4 Conclusions and Future Work

We've presented a robust RED light algorithm which can be parametrized only by the output link bandwidth and admits a simple, efficient implementation at all link speeds. This algorithm shows promise across a range of bandwidths and traffic types.

Although we carried out RED light experiments for two sampling schemes, random samples at 7 ms average and fixed samples at 1.2 ms (MTU-time), the link utilization and drop rates showed no significant differences between the two sampling schemes. There still may be some sample aliasing effects that would affect the fairness, but we leave this for further investigation.

Several items are on our list to be investigated in the future. One is further investigation of sampling, its effects on the control of the queue and efficient sampler implementations. Another is to develop a penalty box algorithm we believe will work well with RED light. We'd like to investigate parameter setting when there are multiple output queues (as for differentiated services), each running a RED. We'd also like to see how the use of TCP-SACK affects results, though it should lead to improved performance for the end systems.

5 References

[1] S. Floyd and V. Jacobson, Random Early Detection Gateways for Congestion Avoidance, IEEE/ACM Transactions on Networking, August 1993.

[2] R. Braden et al., "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC2309, April 1998.

[3] V. Jacobson, "Notes on Using RED for Queue Management and Congestion Avoidance", talk at NANOG 13, ftp://ftp.ee.lbl.gov/talks/vj-nanog-red.pdf, see also http:// www.nanog.org/mtg-9806/agen0698.html, Dearborn, MI, June, 1998.

[4] Sean Doran, RED Experience and Differentiated Queueing, talk at NANOG 13, http:// adm.ebone.net/∼smd/red-1.htm see also http://www.nanog.org/mtg-9806/agen0698.html, Dearborn, MI, June, 1998.

[5] C. Villamizar, and C. Song, "High Performance TCP in ANSNET", Computer Communications Review, V. 24 N. 5, October 1994, pp. 45-60

[6] http://www-nrg.ee.lbl.gov/floyd/red.html

[7] S. Floyd, "RED: Discussions of Setting Parameters" and "Recommendation on Using the 'Gentle_' Variant of RED", November 1997 and March 3, 2000, accessible at http://www.aciri.org/floyd/red.html

[8] V. Jacobson, "Congestion Avoidance and Control", Sigcomm 1988

[9] V. Paxson et. al., "Framework for IP Performance Metrics", RFC 2330, May 1998, p. 21

[10] The simulator ns-2, available at: http://www-mash.cs.berkeley.edu/ns/ and see also the "contributed models" section.

[11] K. Nichols, "Improving Network Simulation with Feedback", Proceedings of LCN '98, October, 1998.

[12] K. Poduri and K. Nichols, "Simulation Studies of Increased Initial TCP Window Size", RFC 2415, September, 1998.

[13] V. Jacobson, reported in "Minutes of the Performance Working Group", Proceedings of teh Cocoa Beach Internet Engineering Task Force, Reston, VA, Corporation for National Research Initiatives, April, 1989.

[14] A. Mankin, "Random Drop Congestion Control", Proceedings of SIGCOMM '90, September 24-27, 1990.