Scalable Configuration of RED Queue Parameters

K. Chandrayana, B. Sikdar and S. Kalyanaraman
Department of Electrical, Computer and Systems Engineering
Rensselaer Polytechnic Institute
Troy, NY 12180 USA
email: {kartikc,bsikdar,shivkuma}@networks.ecse.rpi.edu
Phone: 518 276 8289

Abstract

Active queue management policies and in particular Random Early Detection (RED) are being advocated by the Internet Engineering Task Force (IETF) as additional mechanisms in Internet routers to control congestion and keep network utilization high. RED's performance is highly dependent on the settings of its five control parameters. However, no firm guidelines exist on configuring RED parameters and the current suggestions fail to provide the desired performance over a wide range of scenarios, specially in the presence of many flows. In this paper, we propose a mechanism to configure RED parameters based on evaluating the expected instantaneous length of the queue. We show that by setting the RED parameters min_{th} and max_{th} to lie on either side of this expected queue length, we can ensure that the queue is not underutilized and flows cut their rates before the onset of congestion. This setting also allows the operating point to perform satisfactorily over a wide range of flow counts thereby allowing for a higher degree of scalability. We also show that our proposed mechanism increases the queue goodput, reduces losses and timeouts and increases the fairness when compared to existing guidelines. Our proposals have been verified using extensive simulations.

1 Introduction

Buffering mechanisms are needed in routers to provide protection against losses and complement the endpoint congestion avoidance mechanisms provided by the TCP/IP protocol suite. Traditional buffering techniques are classified as "passive" and include TailDrop queues which accept packets till the maximum queue length is reached and then drop all subsequent packets until the queue length drops below the maximum value. In a recent RFC [1] by the IETF, "active" queue management schemes at the routers were recommended with *random early detection* or RED queues [4] as the suggested buffering scheme. RED queues probabilistically drop incoming packets even though the queue is not full. By dropping packets before the queue fills up, the TCP connections sharing the queue will reduce their transmission rates and ensure that the queue does not overflow.

RED's performance is highly dependent upon it's operating parameters. Though RED is being pushed forward for deployment across the Internet as a means to avoid congestion, guidelines for configuring RED parameters remains an open issue. Recent studies have shown that careful tuning of RED parameters is required to extract the benefits promised by RED and for RED queues to yield performance superior to TailDrop queues [2, 7]. The dependence of the queue performance on the operating point also leads to the problem of scalability. Though a configured RED queue works fine with some flows, when hit with many TCP flows the queue starts behaving like a TailDrop queue.

Though parameter configuration is of utmost importance to RED performance, very little has been published regarding guidelines to setting RED parameters except for some heuristics, which fail to provide desired performance over a wide range of traffic scenarios. The original guidelines for tuning RED queues was presented in [4]. More recent recommendations on setting RED parameters are reported in [5]. Additional guidelines on parameter configuration are presented in [10] and relationships are provided between the loss rates and RED parameters.

In this paper we propose a mechanism to configure RED queues which can also be used to dynamically tune RED parameters based on the current load. Our mechanism is based on setting parameters based on the expected length of a queue without early detection, fed with the same arrival arrival process. Using this expected length to set RED parameters min_{th} and max_{th} ensures that the queue and the output link is not underutilized. It also prevents the sources from increasing their rates too high since the average queue length lies between max_{th} and min_{th} where packets are dropped randomly, forcing TCP sources to cut down their transmission rates. Though RED cannot provide a linear relationship between flow count and average buffer size, our methodology allows us to address the problem of scalability, as the placement of max_{th} and min_{th} between the expected queue length allows it to operate well for a range of flow counts. Our methodology can also be used to dynamically configure RED parameters based on the current traffic conditions as the number of active flows in the traffic changes since our derivations account for the incoming traffic on a per flow level rather that the aggregate traffic. The proposed configuration mechanism performs better than existing guidelines in terms of the queue goodput, loss rates, number of timeouts and the fairness amongst flows.

The rest of the paper is organized as follows. In Section 2 we give a brief description of RED queues and present the existing guidelines for configuring RED queues. We then present our scheme for configuring RED in Section 3. Section 4 presents the simulation results showing the improvement in the performance of RED queue set with our guidelines against those set using guidelines proposed in literature. Finally, Section 5 presents the concluding remarks.

2 Background and Related Work

A solution to the problem of full queues is for routers to drop packets before a queue overflows, so that end nodes can respond to congestion before buffers overflow. By dropping packets before buffers overflow, active queue management allows routers to control when and how many packets to drop. Active queue management schemes have been shown to 1) reduce the number of packets dropped in routers, 2) provide lower-delay interactive services, 3) prevent bias against low bandwidth but highly bursty flows and 4) avoid lock-outs common in TailDrop queues.

Random Early Detection is an active queue management algorithm for routers which aims at providing the Internet the above mentioned performance advantages [4]. In contrast to traditional queue management algorithms which drop packets only when the buffer is full, the RED algorithm drops arriving packets probabilistically. The probability of packet drop increases as the estimated average queue size grows. Note that RED responds to a time-averaged queue length, not the instantaneous one. Thus, if the queue has been mostly empty in the "recent past", RED will not tend to drop packets (unless the queue overflows, of course!). On the other hand, if the queue has recently been relatively full, indicating persistent congestion, newly arriving packets are more likely to be dropped. Transient congestion is accommodated by a temporary increase in the queue length.

RED probabilistically drops packets even before the queue is full based on a weighted average of the total queue length. By using a weighted average (the weighting factor is denoted by w_q), RED tries to avoid over-reactions to bursts and instead reacts to long term trends. The RED queue maintains two thresholds which determine the rate of packet drops: a lower threshold, min_{th}, and an upper threshold, max_{th}. When a packet arrives at the queue, we have three cases for packet drops depending on whether the weighted average queue 1) is below min_{th}, 2) is between min_{th} and max_{th} and 3) greater than max_{th}. If a packet arrives when the weighted queue average is below min_{th} no drop action is taken and the packet is enqueued for transmission. In the second case we have an *early drop* action where the incoming packet is dropped randomly with probability p_b which is calculated as follows. An initial probability $p_b = max_p(avg - min_{th})/(max_{th} - min_{th})$ is first calculated where max_p is a control variable denoting the maximum drop probability and avg is the weighted average queue length. This is used to calculate the actual drop probability based on the number of packets, $count$, enqueued since the last packet was dropped and $p_b = p_b/(1 - count \times p_b)$. In the third case where the weighted average queue length is greater than max_{th}, a *forced drop* occurs and the packet is dropped. Also, the drops when the queue is full but the average queue is less than the maximum queue length, $qlen$ can be considered as forced drops. The reader is referred to [4] for the detailed RED algorithm.

RED offers five control parameters, $qlen$, max_p, min_{th}, max_{th} and w_q to tune RED's dynamics according to requirements. However, the impact of the choice of values of individual parameter on the queue's performance is dependent on the the values of the other too. Thus a judicious choice of parameter values is complicated and it is clear that simple heuristics are not sufficient. We now describe the RED configuration schemes proposed in literature.

2.1 Existing Guidelines on Configuring RED

Rough guidelines for configuring RED were presented in the original RED paper by Floyd and Jacobson [4]. It was suggested that w_q should be set greater than or equal to 0.002 and $min_{th} - max_{th}$ should be sufficiently large to avoid global synchronization. Also, min_{th} should be set sufficiently large to avoid underutilization of the output link. A more recent set of guidelines are presented in [5] which recommends that max_{th} should be three times min_{th}, max_p should be set to 0.1 and w_q should be set to 0.002. The proposal notes that the optimal setting for min_{th} depends on the tradeoff between low average delay and high link utilization and suggest setting it to 5. But various experiments have shown that these parameters setting can still fail, specially when the queue is hit with many flows [11], as would be the case in practice.

In [10], Morris provides a RED configuration mechanism based on the number of flows traversing the queue. He first notes that to avoid excess timeouts, an ideal router for TCP should buffer more than four packets per flow. Then, given that there are n active flows, a maximum desirable loss rate l is chosen which should be less than 3% to allow TCP a window size of four packets. If a TCP keeps w packets in flight, with a round trip time r, the average packets in flight can be calculated as:

$$pif(l,r) = (1 - O(l,r,w(l)))w(l) \tag{1}$$

where, the fraction of time spent in timeout is

$$\begin{aligned} o(l) &= l(1-(1-l)^{13}) \\ O(l,r,w) &= \frac{1}{1+\frac{r}{w(l)\cdot o(l)}} \end{aligned} \tag{2}$$

and the window size $w(l) = \frac{0.87}{\sqrt{(l)}}$. Using the average packets in flight, the $\max_{th}$ can be chosen as

$$max_{th} = \frac{3}{2} \cdot n \cdot pif(l, r). \tag{3}$$

At this point the the choice of $\max_p$ is limited to a value which will keep the queue somewhat less full

$$max_p = \frac{3}{4} l. \tag{4}$$

3 RED Parameter Configuration

Recent studies have shown that the buffering should be proportional to the number of active flows [11]. Further it has been shown that a per-flow buffering of 5-6 packets greatly reduces the timeouts by ensuring that losses are recovered through fast-recovery and fast-retransmit algorithms [11]. Thus the two parameters which determine the performance of RED are $\max_p$ and $\max_{th}$ since they control the amount of buffering available per flow and the rate at which packets are dropped. Note that there is a tradeoff between a conservative/aggressive marking probability and fairness. A conservative marking probability is unfair since a small subset of the competing sources is forced into timeouts while other keep transmitting. On the other hand, an aggressive marking policy is fairer since it distributes the losses more evenly amongst the competing sources. However, an aggressive policy increases the link loss rate thereby decreasing the link utilization. However, if max_{th} is quantified properly, an RED queue can be prevented from behaving like a Tail-Drop and thereby making the choice of max_p easier.

In this section we focus on determining an appropriate value of max_{th} as a function of the number of active flows. Our technique is based on estimating the expected queue length of the RED queue without early and forced drops given a number of flows and setting the RED parameters min_{th} and max_{th} to lie on either side of this expected length. We model the queue as $M^x/M/1/K$ queuing model. The model implies we have "bulk arrivals" (in the form of bursts of packets from the competing TCP sources) of varying sizes to the RED buffer. The interarrival time distribution of the bulks is given by an exponential distribution following the assumptions of [7]. The distribution of packets inside the bulk is modeled by a general distribution $g(x)$. The bulk size depends on the stationary window size distribution of the TCP sources and can vary from 0 to m, where m is dependent on the loss rates and the effect of TCP window limitation and has been characterized in [3, 6, 8, 9]. The processing times of the packets in the router are assumed to be exponentially distributed with mean $1/\mu$ and the offered load to the queue is thus $\rho = E[g(x)]\lambda/\mu$. The buffer size K corresponds to the RED parameter $qlen$ and is the maximum number of packets that the queue can accommodate.

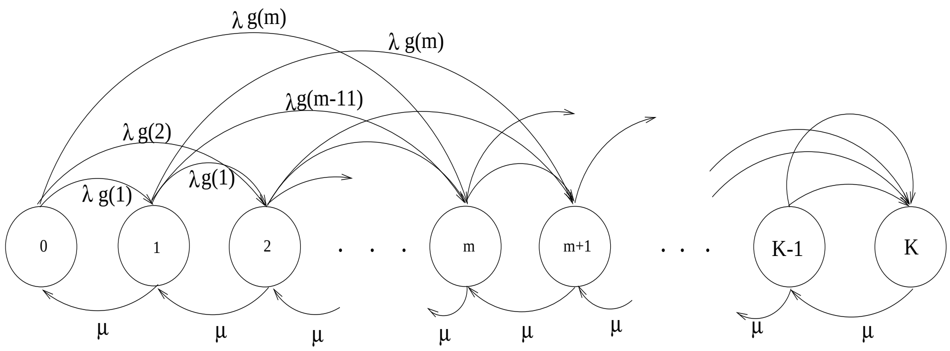


Figure 1: State Diagram for the model

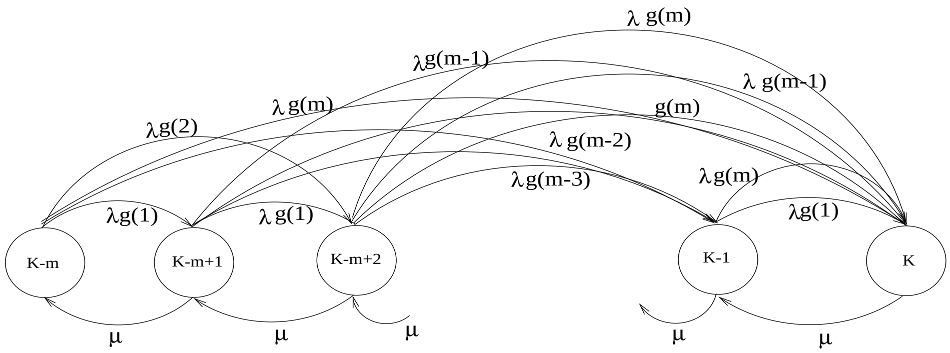


Figure 2: State Diagram for the last node of the model

The weighted average queue in RED is modeled in such a way that it catches up with the instantaneous queue length. This catching up process is governed by exponential weighted moving averages (EWMA) and is slow for small values of w_q. So any approach based on modeling of instantaneous queue length might lead to errors. However, if we find the distribution of instantaneous queue length and base our modeling on its expectation, the model will closely approximate the average queue length thereby reducing any errors. This is an underlying assumption in our work. We assume that if the load is sufficiently high, then at the steady state, the average queue length and the expectation of the instantaneous queue length will be approximately the same.

The Markov chain governing the state-space evolution is shown in Figures 1 and 2. State i denotes that there are i packets in the queue and the largest state in the state-space is governed by the parameter $qlen = K$. The probability that there are i packets in the system is denoted by $p(i)$. Assuming that the service rate is greater than the arrival rate, i.e. $\rho < 1$ (the condition for ergodicity), we may write the steady state equations as

$$\lambda p(0) = \mu p(1) \tag{5}$$

$$(\lambda + \mu)p(r) = \mu p(r+1) + \lambda \sum_{i=1}^{r} p(r-i)g(i), \qquad \forall\ 1 \le r \le m-1 \tag{6}$$

$$(\lambda+\mu)p(r) = \mu p(r+1) + \lambda \sum_{i=1}^{m-1} p(r-m+i)g(m-i), \quad \forall\, m \le r \le K-1 \tag{7}$$

$$\mu p(K) = \sum_{i=0}^{m} p(k-m+i) \sum_{j=0}^{i} g(m-j) \tag{8}$$

with the constraints $\sum_{i=0}^{m} g(i) = 1$ and $\sum_{i=0}^{K} p(i) = 1$.

We use the method of maximum entropy [13] to calculate the probabilities $p(i)$ and the expected queue length (we present only an outline of the method and the final expressions while the details will be available in the final version, if accepted). Consider a system that has a finite or countable set of states S. The prior information regarding the system is described (without loss of generality) in the form of following $I+1$ constraints

$$\sum_{n\in S} p(n) \;=\; 1, \tag{9}$$

$$\sum_{n\in S} f_i(n)p(n) \;=\; < f_i(n) >, \quad 1 \le i \le I, \tag{10}$$

where $\{< f_i(n) >\}$ is a set of expectations defined through appropriate functions of the system states. In general, since I+1<S, there is an infinite number of probability distributions that satisfy these constraints. The maximum entropy distribution can be obtained by the method of the Lagrange's undetermined multipliers leading to the solution [13]

$$p(n) \;=\; \frac{1}{G}\prod_{i=1}^{I} x_i^{f_i(n)}, \tag{11}$$

where $x_i = e^{-\theta_i}$ and θ_i, i=1,2,...,I, are the Lagrangian multipliers determined from the set of constraints, G is the normalizing constant $G = e^{-\theta_0} = \sum_{n\in S}\prod_{i=1}^{I} x_i^{f_i(n)}$ where θ_0 is the Lagrangian multiplier associated with the normalizing constraint. Our system can be characterized using four constraints: 1) normalization ($\sum_{n=0}^{K} p(n) = 1$), 2) utilization factor $\rho = \bar{\lambda}/\mu$ with $\sum_{n=0}^{K} \xi_0(n)p(n) = \rho$ where $\xi_0(n) = \min[1, max(0,n)]$ and $\bar{\lambda}$ is the effective arrival rate, 3) the expected queue length $E[Q] = \sum_{n=0}^{K} np(n)$ and 4) the probability that the queue is full, $p(K)$ which can be written as $\sum_{n=0}^{K} \xi_1(n)p(n) = p(K)$ where $\xi_1(n) = \max[0, n-K+1]$. We can now use Lagrange's method of undetermined multipliers to obtain the set of values for $p(n)$ that maximizes the entropy function $H(p) = -\sum_{n=0}^{K} p(n)logp(n)$ subject to the above four constraints. We get

$$p(n) \;=\; p(0)x^{\xi_0(n)}y^n z^{\xi_1(n)}, \tag{12}$$

where x=$e^{-\theta_1}$, y=$e^{-\theta_2}$, z=$e^{-\theta_3}$, and θ_1, θ_2 and θ_3 are the Lagrangian multipliers corresponding to the constraints 2, 3 and 4. From Equation (12), we can obtain the following recursions

$$p(K) \;=\; zyp(K-1) \tag{13}$$

$$
\begin{aligned}
p(n) &= yp(n-1), \quad 1 \leq n \leq K-1 && (14)\\
p(1) &= xyp(0) && (15)
\end{aligned}
$$

The quantities x, y and z can be calculated by substituting Equation (12) into the steady state equations and we have

$$
\begin{aligned}
x \cdot y &= \frac{\lambda}{\mu} && (16)\\
(\lambda + \mu) &= \mu y + \lambda \sum_{i=1}^{r} y^{-i} g(i) && (17)\\
z &= \sum_{i=0}^{m} y^{i-m-1} \sum_{j=0}^{i} g(m-j). && (18)
\end{aligned}
$$

The above six equations define the solution to our queueing model. Note that the solutions are generic in nature and are valid for any arbitrary choice of the burst or window size distribution $g(x)$. Distributions to characterize $g(x)$ for TCP sources and to estimate λ from the number of active sources is presented later in the section.

3.1 Parameter Settings

Let the function $f(\lambda, \bar{g})$ denote the expectation of the instantaneous queue length in terms of the arrival rate λ and the bulk size distribution *g(x)*. The expected value of instantaneous queue will be in terms of the arrival rate and the expectation of distribution of the bulk size and is given by

$$
f(\lambda, \bar{g}) = E[Q] = \sum_{i=1}^{K} ip(i) \tag{19}
$$

where $\bar{g} = E[g(x)]$. To configure the RED parameters min_{th} and max_{th}, we propose placing them such that $E[Q]$ lies between them and is equidistant from from both min_{th} and max_{th}, i.e.,

$$
\frac{max_{th} - min_{th}}{2} + min_{th} = E[Q] \tag{20}
$$

This allows scalability of the parameter settings as the number of flows in the queue changes. For moderate increase in the number of flows, the expected queue length would still be less than max_{th} preventing the RED queue from behaving like a TailDrop while keeping the utilization high. On the other hand, if the number of active flows decreases, the queue length reduces, allowing the sources to increase their rates without causing congestion since the queue has been configured for a larger number of flows. Now, using the guideline $max_{th} = 3min_{th}$ [5] we can write min_{th} and max_{th} in terms of $E[Q]$ as

$$
min_{th} = \frac{E[Q]}{2} \quad \text{and} \quad max_{th} = \frac{3E[Q]}{2} \tag{21}
$$

The remaining three parameter settings are quite straightforward. The maximum queue length $qlen$ is determined by the physical system configuration and is the size of the queue. The maximum loss rate max_p is set according to the loss rate conditions prevailing in the network which should be around 2% under normal conditions and thus good guideline would be to set max_p between 0.05 and 0.1. The parameter w_q determines how fast the weighted average responds to the instantaneous queue lengths. In [4] quantitative guidelines for w_q are presented, in terms of the size of the transient burst that the queue can accommodate without dropping any packets at all and in general should be less than 0.005 with 0.002 being the default.

An RED queue starts behaving as TailDrop when the weighted average queue reaches max_{th} and all packets experience a forced drop. A rough estimate of the input arrival rate at which the expected queue length exceeds max_{th} and thus the range of the number of flows over which the parameter setting is valid can then be obtained by solving Equation 19 with respect to the arrival rate. So, the worst case scenario for the RED (when it becomes Tail-Drop) can be written as

$$\lambda \quad = \quad F(max_{th}, \bar{g}) \tag{22}$$

where F is the inverse mapping of the function f.

3.2 Arrival Rate as a Function of the Number of Flows

We now address the issue of estimating the batch arrival rate λ as a function of the number of active flows. We assume that the reader is familiar with the basic mechanisms of TCP like slow-start, timeouts, fast-retransmit and recovery etc. and is referred to [14] for further details. For simplicity, we consider the case when all the sources have the same round trip time (RTT). The case for heterogeneous RTTs is similar in nature. TCP transmits its packets according to a window based flow control mechanism and transmits a window's worth of packets every RTT. Thus the batch arrival rate corresponding to each TCP source is $1/RTT$. However, when a TCP source encounters a loss and goes into a timeout, no packets are sent till the retransmission timer expires. If we denote the fraction of time a TCP source spends in timeout by γ, the effective batch arrival rate corresponding to each source is thus $(1-\gamma)/RTT$. Now, if there are N flows in the queue, the total batch arrival rate is given by

$$\lambda = \frac{1-\gamma}{RTT}N \tag{23}$$

We use the derivations of [12] to estimate the fraction of time a TCP source spends in the timeout phase. We first note that losses in TCP can be recovered using either a timeout or a fast retransmit. A TCP flow can then be considered as a sequence of timeout and congestion avoidance phases (a fast retransmit is followed by the congestion avoidance phase) and we need to find their expected durations to obtain γ. From [12], a TCP source experiencing a loss rate of p has an expected timeout duration $E[TO]$ given by

$$E[TO] = T_O \frac{1 + p + 2p^2 + 4p^3 + 8p^4 + 16p^5 + 32p^6}{1 - p} \tag{24}$$

where T_O is the period of time a sender waits before retransmitting an unacknowledged packet. We denote the probability that an arbitrary loss leads to a timeout by Q and the expected duration of a congestion avoidance phase (where packets are transmitted every RTT) is denoted by $E[CA]$. Q and $E[CA]$ can be expressed as

$$Q = \min\left(1, \frac{(1 + (1-p)^3(1 - (1-p)^{E[W]-3}))(1 - (1-p)^3)}{(1 - (1-p)^{E[W]})}\right) \tag{25}$$

$$E[CA] = \begin{cases} \left(\frac{b}{2}E[W_u] + 1\right) RTT & \text{if } E[W_u] < W_{max} \\ \left(\frac{b}{8}W_{max} + \frac{1-p}{pW_{max}} + 2\right) RTT & \text{otherwise} \end{cases} \tag{26}$$

where $E[W] = \min(W_{max}, E[W_u])$ is the expected value of the window size, $E[W_u] = \frac{2+b}{3b} + \sqrt{\frac{8(1-p)}{3bp} + \left(\frac{2+b}{3b}\right)^2}$ is the expected value of the unconstrained window size and W_{max} is the receiver's advertised maximum window size. The fraction of the time the TCP source spends in timeout is then

$$\gamma = \frac{QE[TO]}{E[CA] + QE[TO]} \tag{27}$$

3.3 Window Size Distribution

The window size distribution of TCP flows in ideal congestion avoidance (TCP flows without timeouts) has been investigated in [6] and it was shown that the average window size is proportional to $1/\sqrt{p}$ where p is the loss rate experienced by the flow. A similar result has also been obtained in [10] using rough calculations which show that the average window size is given by $0.87/\sqrt{p}$. Misra and Ott [8] extend the analysis of idealized TCP connections for the cases of variable packet loss rates for RED like queues. In [9] this model is extended to model the window size distribution for multiple persistent flows.

While the models described above are obtained using analytic derivations using a number of simplifying assumptions, the window size distribution of TCP flows based on empirical measurements is presented in [3]. Using measurements conducted on a bottleneck link connecting two corporate LANs it was shown that the TCP congestion window size can be approximated by a truncated normal distribution (i.e. with no negative values). The window distribution $g(x)$ used in the derivations of this section and of Section 3 can be characterized using any of the models mentioned above. In our simulations, we use the truncated Gaussian distribution of [3] as the distribution for $g(x)$.

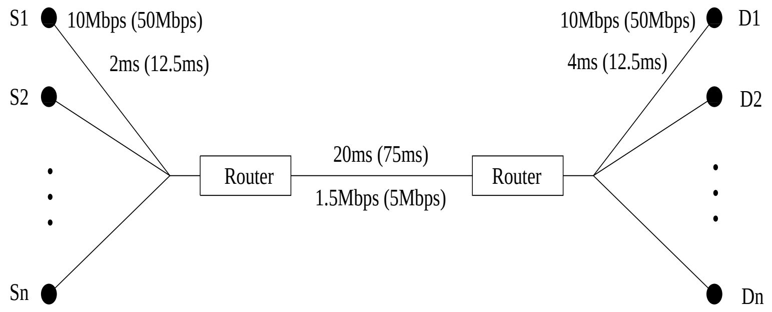


Figure 3: Topology used in the experiments. The figure shows values for two sets of experiments with different link rates with the second set of values in the parentheses

4 Simulation Results

To verify our results, we carried out extensive simulations using the network simulation *ns* [15]. Due to space restrictions we present the results for only two sets of simulations and the others are similar. The topology used for these simulations is shown in Figure 3 where we have a number of TCP sources passing through a bottleneck link where the router deployed a RED queue. The two set of experiments reported here have the same topology though the link parameters differ and are marked in the figure. The TCP sources are of the TCP Reno family and do not employ delayed acknowledgments.

In the first set of experiments, we configure the RED queue for 48 TCP flows each with an RTT of $52ms$. Using the expressions of the previous chapter and the link rates and propagation times shown in Figure 3, $E[Q]$ was calculated to be 79.54. $g(x)$ had a truncated Gaussian distribution with mean 4, variance 4 and the maximum burst size of 8. Following the guidelines of the previous section, the RED parameters were set to $min_{th} = 40$, $max_{th} = 120$, $max_p = 0.10$, $w_q = 0.002$ and the queue length $qlen = 200$. Figure 4(a) shows the simulation results for the instantaneous and average queue length of the RED queue set with these values and 48 sources. Note that our derivations predict the expected queue length quite closely and max_{th} is configured high enough to prevent forced drops. Figures 4(b) and 4(c) show the simulation results for the RED queue configured with the same values but with 32 and 64 flows. Note than even with a 33% change in the number of flows, our configuration keeps the average queue length less than the configured max_{th} showing the scalability of our parameter configuration. Figure 4 also shows the results for an RED queue configured with the guidelines in [10]. Following the guidelines of [10], $min_{th} = 20$ and $max_{th} = 60$ with the other parameters being the same. Note that this scheme fails to keep the queue from behaving as a TailDrop queue and the average queue stays at 60 most of the time, for all three cases. Also, note that the configuration is not scalable as is evident from the graphs for 32

No. of	Goodput (pkts/sec)		Loss Rate (%)		Timeouts		Fairness	
Flows	Proposed	Morris	Proposed	Morris	Proposed	Morris	Proposed	Morris
32	77.71	77.78	5.17	6.35	366	840	0.10	0.19
48	51.90	52.00	5.72	6.05	642	1472	0.20	0.27
64	39.10	39.10	6.11	5.88	1113	1861	0.25	0.33

Table 1: Comparison of throughput, loss rates, timeouts and the fairness of our proposed configuration and that proposed in [10].

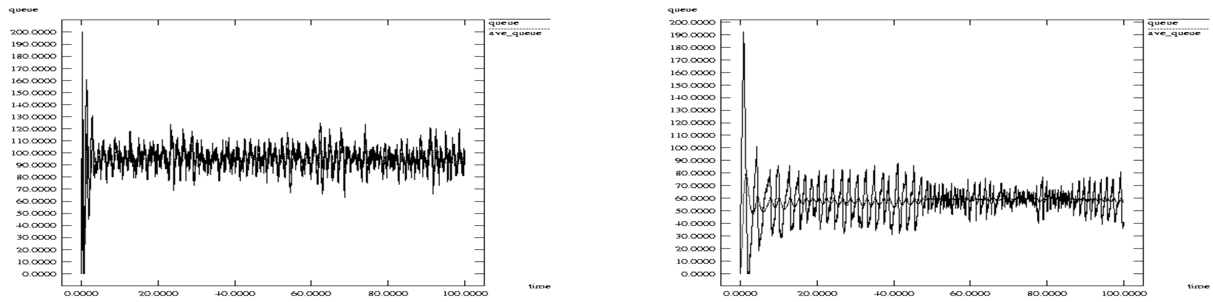
and 64 sources in Figure 4. In Table 1 we compare the performance of the queue with our settings with those from [10] in terms of the goodput, the drop rates experienced by the flows, number of timeouts experienced by the flows and the fairness. We define fairness as the coefficient of variation of the throughputs of the various sources. Thus a lower value for the fairness index implies a fairer queue. Note that our configuration performs better than that of [10] in almost all cases.

Figure 5 shows the simulations for the second set of parameters for the topology in Figure 3 corresponding to a RTT of $200ms$. For this topology, we configured the RED queue of size $qlen = 1000$ for 150 sources with $g(x)$ given by a truncated Gaussian distribution with mean 4.4, variance 4 and maximum burst size 9. The expected instantaneous queue length $E[Q]$ was calculated to be 592.01 and the queue parameters were set at $min_{th} = 300$ and $max_{th} = 900$ with $max_p = 0.1$ and $w_q = 0.002$. The simulation results for the queue configured with these parameters is shown in Figure 5 (a). Figures 5 (b) and (c) show the simulation results for the same configuration but for 100 and 200 flows respectively. Again we note that even though the number of flows has large variations, our configuration is able to scale and prevent the sources from transmitting too fast. As in the previous case, we also conducted a comparison of our scheme with the configuration according to [10] (with $min_{th} = 136$ and $max_{th} = 408$) which are presented in Table 2. The results are similar to those for the previous case with the proposed configuration outperforming that from [10].

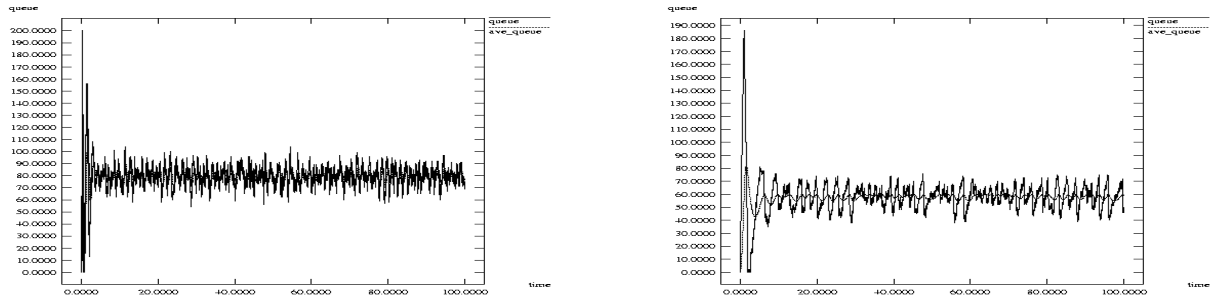
Please note that the proposed value of $min_{th} = 5$ and $max_{th} = 20$ in [5] fail to be practical even for moderate loads. Hence, we do not present any comparison with these guidelines.

5 Discussion and Conclusions

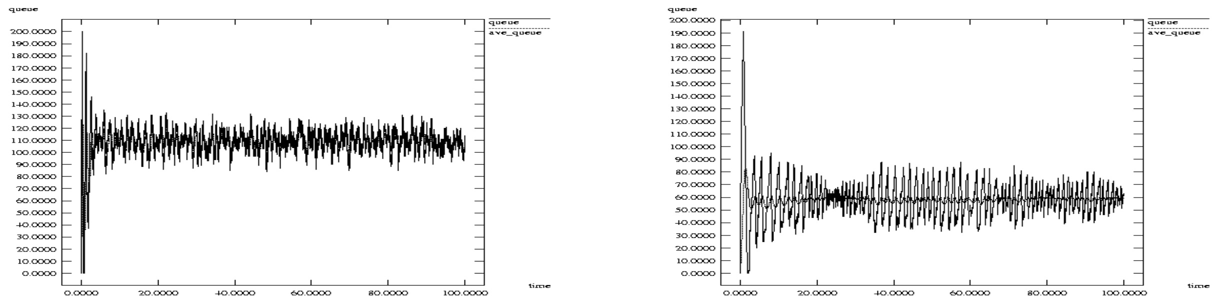
In this paper we presented a scalable mechanism to configure RED queue parameters. While the choice of RED parameters has a profound impact on the behavior of the queue, parameter



(a) 48 flows

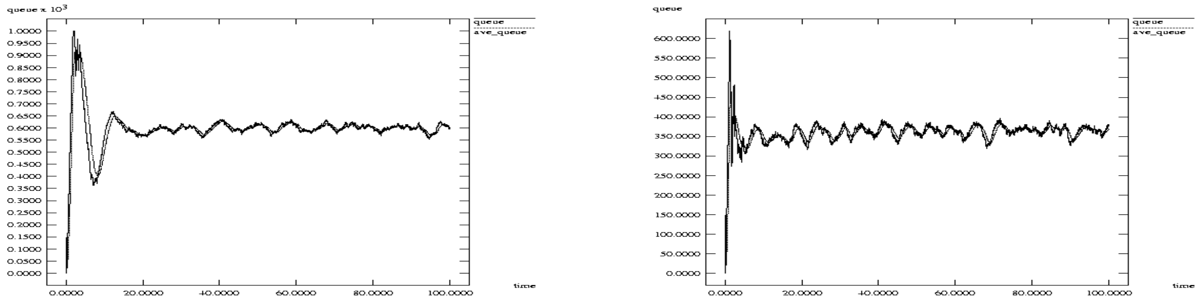


(b) 32 flows

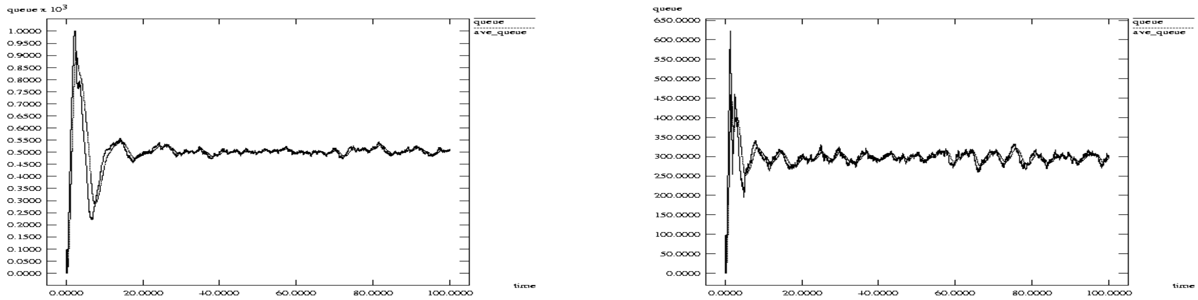


(a) 64 flows

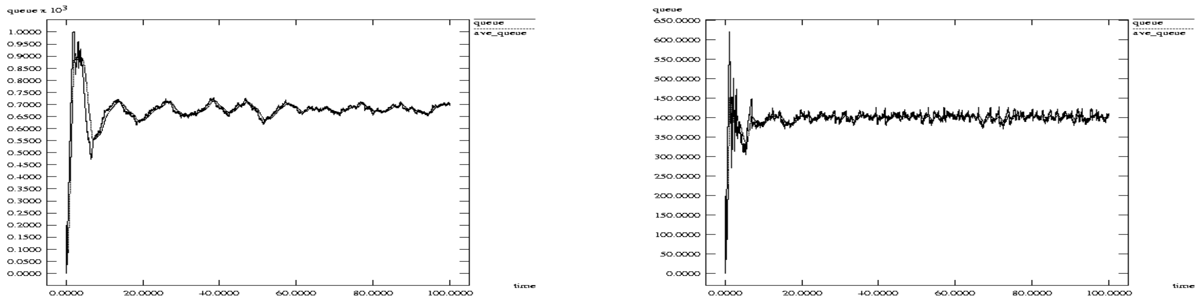
Figure 4: Instantaneous and average queue lengths for an RED queue configured for 48 flows and fed with 32, 48 and 64 flows for the proposed configuration (left) and that proposed in [10] (right).



(a) 150 flows



(b) 100 flows



(a) 200 flows

Figure 5: Instantaneous and average queue lengths for an RED queue configured for 150 flows and fed with 100, 150 and 200 flows for the proposed configuration (left) and that proposed in [10] (right).

No. of	Goodput (pkts/sec)		Loss Rate (%)		Timeouts		Fairness	
Flows	Proposed	Morris	Proposed	Morris	Proposed	Morris	Proposed	Morris
100	82.73	82.95	2.29	4.09	288	645	0.15	0.14
150	55.31	55.53	3.39	5.77	585	1297	0.16	0.15
200	41.58	41.81	4.34	6.67	863	2205	0.18	0.17

Table 2: Comparison of throughput, loss rates, timeouts and the fairness of our proposed configuration and that proposed in [10].

configuration has remained an open area and existing guidelines fail to provide satisfactory performance over a large number of scenarios. Our method configures the RED parameters based on the expected queue length of a DropTail queue fed with the same arrival process. By setting the key parameters min_{th} and max_{th} on either side of this expected queue length, we ensure that the output link remains utilized while effectively controlling the rates of the flows preventing the onset of congestion. This also provides scalability to the configuration in the face of changing flow conditions. Our results show that even for large variations in the number of flows (> 35 %) the expected queue length stays below the configured max_{th} preventing the RED queue from behaving like a DropTail at all times.

Our configuration policy also performs favorably when compared to the other mechanisms proposed in literature. Comparisons were made in terms of four parameters: goodput, loss rates, timeouts and fairness and our scheme outperforms that of [10] in almost all cases. Our configuration is able to reduce the loss rates by keeping the RED queue in the random drop part most of the time thereby preventing the forced drops. Also, the reduction in the timeouts is a direct consequence of the queue staying in the random drop part. It is well known that TCP is able to recover individual losses (which result from random drops) without timeouts but in the presence of bursty losses of a TailDrop queue, TCP resorts to timeouts [12]. The increase in the fairness can also be tied to the reduction in timeouts.

References

[1] B. 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, L. Zhang, "Recommendations on queue management and congestion avoidance in the Internet," *RFC-2309,* April 1998.

[2] M. Christiansen, K. Jeffay, D. Ott and F. D. Smith, "Tuning RED for web traffic," *Proc. of ACM SIGCOMM,* pp. 139-150, Stockholm, Sweden, September 2000.

[3] T. Éltető and S. Molnár, "On the distribution of round-trip delays in TCP/IP networks," Proc. of IEEE Conference on Local Computer Networks, Lowell, MA, October 1999.

[4] S. Floyd and V. Jacobson, "Random early detection gateways for TCP congestion avoidance," *IEEE/ACM Transactions on Networking* vol. 1, no. 4, pp. 397-413, August 1993.

[5] S. Floyd, "RED: Discussions of setting parameters," `http://www.aciri.org/floyd/REDparameters.txt`, November 1997.

[6] M. Mathis, J. Semke, J. Mahdavi and T. Ott, "The macroscopic behavior of the TCP congestion Avoidance Algorithm," *Computer Communications Review,* Vol. 27, No. 3, pp 67-82, July 1997.

[7] M. May, T. Bonald and J.-C. Bolot, "Analytic evaluation of RED performance," *Proc. of IEEE INFOCOM,* pp. 1415-1424, Tel-Aviv, Israel, March 2000.

[8] A. Misra and T. J. Ott, "The window distribution of idealized TCP congestion avoidance with variable packet loss," *Proc. of IEEE INFOCOM,* pp. 1564-1572, New York, NY, March 1999.

[9] A. Misra, T. J. Ott and J. Baras, "The window distribution of multiple TCPs with random loss queues," *Proc. of IEEE GLOBECOM,* pp. 1714-1726, Rio de Jenerio, Brazil, December 1999.

[10] R. Morris, "Scalable TCP congestion control ," Ph.D. Thesis, Department of Computer Science, Harvard University, Boston, MA, January 1999.

[11] R. Morris, "Scalable TCP Congestion Control," *Proc. of IEEE INFOCOM,* pp. 1176-1183, Tel-Aviv, Israel, April 2000.

[12] J. Padhye, V. Firoiu, D. Towsley and J. Kurose, "A simple model and its empirical validation," *Proc. of ACM SIGCOMM'98,* Vancouver, BC, Canada, pp 303-314, September 1998.

[13] H. G. Perros, *Queueing Networks with Blocking: Exact and Approximate Solutions,* Oxford University Press, 1994.

[14] W. R. Stevens, *TCP/IP Illustrated Volume 1,* Addison Wesley, 1994.

[15] UCB/LBLN/VINT Network Simulator - *ns* (version 2), `http://www.isi.edu/nsnam/ns`, 1997.