TECHNICAL RESEARCH REPORT

Balanced-RED: An Algorithm to Achieve Fairness in the Internet

by Farooq M. Anjum and Leandros Tassiulas

CSHCN T.R. 99-9
(ISR T.R. 99-17)

The Center for Satellite and Hybrid Communication Networks is a NASA-sponsored Commercial Space Center also supported by the Department of Defense (DOD), industry, the State of Maryland, the University of Maryland and the Institute for Systems Research. This document is a technical report in the CSHCN series originating at the University of Maryland.

Web site http://www.isr.umd.edu/CSHCN/

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
1999		**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Balaanced-RED: An Algorithm to Achieve Fairness in the Internet	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Air Force Office of Scientific Research,875 North Randolph Street,Arlington,VA,22203-1768**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**31**	

Balanced-RED: An Algorithm to achieve Fairness in the Internet *†

Farooq M. Anjum and Leandros Tassiulas

Dept. of Electrical Engineering and Institute for Systems Research

University of Maryland at College Park

College Park, MD 20742

March 8, 1999

ABSTRACT

The problem of fair bandwidth sharing among adaptive (TCP) and non-adaptive (i.e. CBR-UDP) flows at an Internet gateway is considered. An algorithm that drops packet preventively, in an attempt to actively penalize the non-adaptive traffic that attempts to "steal" buffer space, and therefore bandwidth from the adaptive traffic flows, is presented. The algorithm maintains minimal flow state information and is therefore scalable. The performance of the algorithm is compared with other gateway algorithms and it is shown that, in the presence of non-adaptive traffic, it achieves a more balanced bandwidth allocation among the different flows. The behavior of a flow subjected to the given algorithm has also been analyzed in detail.

*This research was supported in part by NSF under a CAREER award NCR-9502614 and by the AFOSR under grant 95-1-0061.

†A partial version of this paper appeared in Infocom 99

1 Introduction

The major transport protocol in use over the Internet is the TCP that provides end-to-end congestion control. The way TCP works is that it keeps increasing the sending rate of packets as long as no packets are lost. As the network becomes congested and the buffers at the gateways fill up, packet losses occur. In response to that, TCP decreases the sending rate. Thus the sending rate of the adaptive applications is changed according to the level of congestion perceived in the network. In this manner TCP adjusts the long term transmission rate without any need for feedback from the network.

With the proliferation of applications and users, it is no longer possible to rely exclusively on the sources to implement end-to-end congestion control. In fact, many applications like Internet telephony which are expected to be widely used do not have these congestion control mechanisms and are therefore non-adaptive. The non-adaptive applications have no way of inferring and hence reacting to the congestion in the network. When the network consists of both adaptive and non-adaptive traffic, the packets for both compete for buffer space at the gateway. As congestion builds up, packets for both types of applications may be dropped. In response to this, the adaptive applications decrease their sending rate while the non-adaptive applications may not change their sending rate of packets. As a result, the adaptive applications are penalized. While issues of unfairness to the adaptive traffic do arise, a major resulting problem is that this sort of behavior acts as a disincentive for the deployment of applications incorporating end-to-end congestion control mechanisms. This drives the network to undesirable, congestion dominated operating regimes, characterized by large number of packet drops in the network with no useful work being done.

This possibility calls for designing mechanisms within the network which provide an incentive for applications to have end-to-end feedback. The best place to provide such mechanisms is obviously the gateway since the different flows interact at the gateway. An approach for the gateway would then be to keep a separate queue for each flow and use mechanisms such as round robin scheduling [8] over the different queues. This approach not only ensures fairness, but also acts as an incentive for applications to have end-to-end congestion control. Another approach is for the gateways to

provide feedback to the sources indicating how much data they are allowed to send [1]. But a main problem with these approaches is that either they do not scale well considering that many short-lived flows pass through an Internet gateway or they are not in conformance with the current design principles of the Internet.

In this paper we provide a gateway algorithm that provides fairness to the different flows passing through a gateway irrespective of whether the flow incorporates end-to-end feedback or not. Furthermore, the algorithm conforms with the scaling and design principles of the Internet. The resulting advantage is that non-adaptive applications have no incentive to remain so. The characteristics of the algorithm provided, Balanced Random Early Detection (BRED), is that it is simple to implement, maintains a small state and is hence scalable. It should be remarked that a flow in this paper is identified by its source/destination addresses.

The plan of the paper is as follows. In section 2, we look at the gateway algorithms proposed or currently in use and point out the problems associated with these. The proposed algorithm BRED, is given in section 3. In section 4, we analyze the behavior of both adaptive and non-adaptive flows when subjected to the BRED gateway algorithm. Apart from leading to a better understanding of the interaction between BRED and the flows using the gateway, the analysis also offers heuristic guidelines for setting the different parameters in BRED. In section 5 we present simulation results demonstrating and comparing the performance of BRED with the other gateway algorithms.

2 Gateway Algorithms

The gateway buffer management approach currently, in almost universal use is the drop tail gateway in which packets arriving to a full buffer are dropped. While simple to implement, these gateways have many drawbacks associated with them [6] such as causing systematic discrimination against some connections, being biased against bursty traffic etc. A variation on the drop-tail strategy is the drop-front strategy [13]. In case of drop front strategy, when packets arrive at a full buffer the arriving packets are accepted but the packets at the start of the buffer are dropped.

Random Drop gateways on the other hand, randomly choose a packet to drop from the gateway queue when an arriving packet finds the queue full. In Early Random Drop (ERD) gateways [3], if the queue length exceeds a certain drop level, then the gateway drops each packet arriving at the gateway with a fixed drop probability. But both these algorithms are not successful in controlling misbehaving users since they do not differentiate between the adaptive and non-adaptive flows. Random Early Detection (RED) gateways [7], on the other hand, were designed to address the shortcomings of drop-tail gateways and also to provide a congestion avoidance mechanism. The way RED works is that it calculates the average queue size using a low-pass filter with an exponentially weighted moving average. The average queue size is compared to a minimum and maximum threshold. When the average queue size is less than the minimum threshold, no arriving packets are dropped. When the average queue size is greater than the maximum threshold, all arriving packets are dropped. When the average size of the queue is between the two thresholds the probability of dropping a packet is a function of the average queue size. While RED gateways have many advantages, they do not attempt to ensure fairness in terms of the bandwidth received by each flow nor do they explicitly control misbehaving users. Dropping packets from flows in proportion to their bandwidths as done in RED does not always lead to fair bandwidth sharing [9]. Hence, additional mechanisms have to be added to RED gateways to prevent the misbehaving non-adaptive traffic from gaining at the expense of adaptive traffic.

Currently, many studies have been concerned with changing RED at the routers [9, 5] so as to make it fairer in the sense that, the different flows passing through a RED gateway get an approximately equal share of the bandwidth of the backbone link. The approach taken in [5] is to identify the misbehaving users at a RED gateway and subject them to different treatment using the help of scheduling mechanisms. But, the problem of identification of misbehaving users is not properly set-up. On the other hand, FRED [9] suggests changes to the RED algorithm to ensure fairness amongst all the flows. The approach proposed here is to maintain minimum and maximum limits on the packets that a flow can have in the buffer. Flows which consistently violate the maximum limits are marked and subjected to more aggressive dropping. But it has been seen that FRED frequently fails to achieve fair share for the flows in many cases.

Another approach different from RED is the longest queue drop (LQD) gateway [11]. With a LQD gateway, whenever the buffer is full, a packet of the flow with maximum number of packets in the buffer is dropped such that the chosen packet is farthest from the end of the buffer. But as a result of this the cbr flows are penalized very much when their input rate becomes far larger than their fair share. This is due to the nature of dropping from front. Dropping packets from the front was proposed in [11] since it would lead to faster detection of packet drops by the tcp flows but this leads to a deterioration in the behavior of the non-tcp flows. Thus under this approach the gateways have no control over the degree of penalization that a non-conforming non-adaptive flow is subjected to. Other algorithms similar to LQD are also proposed in [11] such as approximated longest queue drop (ALQD) and random LQD (RND). LQD gateways and its variants are complex to implement, cause buffer overflows and also act as a congestion control mechanism and not a congestion avoidance mechanism. Note that by allowing buffer overflows, schemes such as Explicit Congestion Notification [4] cannot be taken advantage of. Hence, the problem of designing an effective gateway algorithm to deal with both adaptive and non-adaptive traffic while providing incentives for flows to incorporate end-to-end feedback is still unsolved. This problem is complicated by the fact that the gateway has to deal with traffic sources of different durations also.

BRED, the algorithm proposed in this paper tries to regulate the bandwidth of a flow by doing per flow accounting for the buffer active flows. Drop or accept decision for a packet is then based on the buffer occupancy of the flow. The extra complexity introduced at the gateway is proportional to the buffer size since state is maintained only for the flows having packets in the buffer. Note that the concept of per active flow accounting has also been proposed elsewhere [9, 11]. As we see later in the section on simulation study, BRED is very effective in ensuring fair bandwidth division amongst the adaptive and non-adaptive flows.

3 BRED

We consider flows, both adaptive and non-adaptive, from different sources passing through the gateway and sharing the buffer. The packets in this buffer are assumed to be dequeued on a FCFS basis. The goal at this point is to come up with an algorithm according to which the congested gateway performs, by making drop or accept decision on an incoming packet, such that fairness to different flows is ensured. Towards this end we propose the algorithm BRED, which is described next.

BRED maintains a variable $\mathbf{qlen}_i$, which is a measure of the number of flow i packets in the buffer, for each flow having packets in the buffer. When a packet from a flow arrives, decision to drop or accept the packet is based on the number of packets that the given flow already has in the buffer. By keeping state just for the buffer active flows, the extra information needed to be maintained by the gateway in the worst case scenario is proportional to the buffer size. If the different flows have different packets sizes then the algorithm would have to be used in the byte mode and not in the packet mode as given. Regarding the notation, B denotes the buffer size while variable $Nactive$ is a measure of the number of flows having packets in the buffer. The algorithm is as follows:

Parameters

1. l_1 : minimum number of packets that a flow can have in the buffer before its packets start being dropped with probability p_1
2. l_2 : number of packets that a flow can have in the buffer before its packets start being dropped more aggressively with probability p_2 with $p_2 > p_1$
3. W_m :maximum number of packets that the flow is allowed to have in the buffer

For each arriving packet from flow i

1. Initialize the flow state if state not already present. Flow state $\mathbf{qlen}_i$ is zero at initialization. If flow state is zero, increment $Nactive$

2. if $\mathbf{qlen}_i \geq W_m$ **OR** buffer overflows drop packet and **return**

3. if $W_m >$**qlen**$_i > \mathrm{l}_2$ drop packet with probability p_2 and **return**

4. if $l_2 \geq$**qlen**$_i \geq \mathrm{l}_1$ drop packet with probability p_1 and **return**

5. if $\mathbf{qlen}_i < \mathrm{l}_1$ accept packet and **return**

6. Increase $\mathbf{qlen}_i$ if packet accepted

For each departing packet from flow i

1. Decrement $\mathbf{qlen}_i$. If $\mathbf{qlen}_i$ is zero decrement $Nactive$

In the next section, we analyze the behavior of both adaptive and non-adaptive flows passing through a BRED gateway where we also give guidelines as to the selection of the different parameters. Note that because of the TCP behavior it may not be possible for every flow to have the same bandwidth share. The goal is to minimize the differences in the bandwidth achieved by each flow.

4 Performance Analysis

As can be seen from the description of BRED, there are five control parameters namely W_m, l_1, l_2, p_1 and p_2. In this section our goal is to study the effects of these control variables on a flow so as to obtain guidelines on setting these variables in practice. Hence, we analyze the behavior of TCP and CBR flows passing through a BRED gateway. The analysis of TCP flows is based on the concept of packet trains which we introduce next.

We now consider the packet train model. A Tahoe transmitter when sending new data is generally either in the slow start phase or the congestion avoidance phase. The loss of a packet in either of these phases is subsequently followed by mechanisms to recover the lost packet(s). A Tahoe sender uses either the fast retransmit mechanism whereby the arrival of a certain number of duplicate acks signals a packet loss or the retransmit timeout mechanism whereby the non-arrival of the packet

acknowledgment within a certain time interval is taken to signify the packet loss. The sender resorts to slow-start after the detection of a packet loss by reducing the window size to one. Hence, the operation of a TCP transmitter can be considered in terms of cycles. A cycle of a Tahoe sender starts with the slow start phase and ends following the detection of a packet loss either on the basis of the ack based fast retransmit mechanism or the timer based retransmit timeout mechanism. Thus, every cycle can be considered to have two stages, the first stage in which the sender is either in the slow start or the congestion avoidance phase. The second stage consists of the retransmit timeout interval. While the first stage is present in every cycle, the second stage may or may not be present depending on whether enough packets are successful after the lost packet for the fast retransmit mechanism to succeed. Hence, it can be seen that the window size increases during a cycle since there are no packet losses in a cycle and window size decreases between cycles.

In order to explain the working of Tahoe, we define kth minicycle of the first stage of ith cycle to be the time taken to transfer the kth window of packets during the first stage of the ith cycle. Hence, the first minicycle corresponds to the first window of packets on the start of a new cycle. Thus, every cycle of TCP if in slow start phase begins with one packet being transmitted in the first mini-cycle and if it is in the congestion avoidance phase, then the number of packets transmitted in the first mini-cycle depends on the window size when the packet drop was detected in the previous cycle. In every successive mini-cycle the number of packets transferred is double the number of packets transferred during the present mini-cycle as long as TCP is in the slow start phase. During the congestion avoidance phase the number of packets transferred in each mini-cycle is one more than the number of packets transferred during the previous mini-cycle.* This goes on until there are one or more packet drops in a particular cycle causing the ack cycle to either dry up or generate enough duplicate acks resulting in the end of the first stage of the cycle. Thus a cycle consists of a collection of contiguous mini-cycles such that no packet drops are detected by the sender between the start of the first mini-cycle of the cycle and the end of the last mini-cycle of the same cycle. A typical cycle of Tahoe is illustrated in figure 1 with the notation being clarified later.

*For ease of explanation, we are ignoring some constraints like delayed acks, which though can be easily incorporated into the description

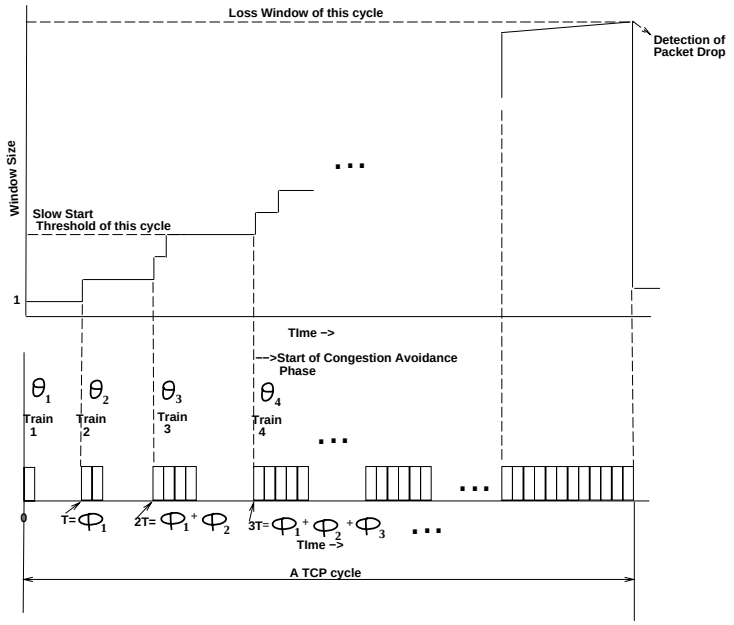

Figure 1: *Illustration of packet trains during a cycle of Tahoe*

We assume that all the packets in a mini-cycle travel in what we call a train. Thus, there is a packet train in every mini-cycle and the size of the packet train in the kth mini-cycle $k > 1$, depends on the size of the packet train in the previous mini-cycle and the phase of TCP. A new packet train starts once the ack for the first successful packet of the previous packet train comes back. This train ends when the packets corresponding to the last ack of the previous train have been transmitted by the sender or a certain number of duplicate acks reach the sender thus giving some length to the train. Start of a successful timeout at any point also terminates a train. The length of the packet train which is the distance between the first packet of the train and the last packet of the train keeps on increasing since the number of packets in successive trains is an increasing function. A great convenience offered by the packet train concept is that it helps to differentiate packets on the basis of the mini-cycle that they belong to. This as we see later greatly helps in calculating the throughput of a flow. This is because once we know the number of trains in a cycle as well as the window size σ when the packet drop was detected in the last cycle, the expected number of packets in the cycle as well as the mean cycle duration can be easily calculated. As we see later, this is the approach that we take to characterize the throughput of a flow.

4.1 Single Congested Node–TCP flow

Now, consider a BRED gateway with many TCP flows passing through it. We concentrate initially on a single TCP flow such that the other flows do not affect this flow. This can be ensured if the buffer occupancy of this flow is not affected by the other flows. A simple way to achieve this is by having no constraint on the buffer size. Of course, this is an unrealistic assumption which we remove in the next subsection. Thus, when a packet of the flow of interest reaches the gateway, it is accepted if there are less than l_1 packets at the gateway. Hence, packets of the flow are dropped only when the number of packets of the flow in the buffer is greater than l_1. Further, if there be between l_1 and l_2 packets of the flow at the gateway, then a packet can be dropped with probability p_1 and if between l_2 and W_m packets, then a packet can be dropped with probability p_2. Note that if a flow has l_1 packets at the buffer then the window size corresponding to the flow at that time necessarily has to be greater than or equal to l_1. For ease of analysis, we assume that the window size equals the number of packets of the flow in the buffer. This leads to a conservative analysis of the TCP flows which should be okay as it is advantageous to the adaptive TCP flows. The main purpose of this analysis is to study how the behavior of a TCP flow varies as a function of the drop thresholds and the drop probabilities so that heuristic guidelines to setting these values in the algorithm provided are obtained. To this end, we next concentrate on characterizing the throughput of the TCP flow under the above scenario at the gateway.

Consider the packet train model. Let the number of packets transferred during a cycle be denoted by $\hat{Q}$ while the duration of the cycle be τ. Now in order to determine the goodput η obtained by the TCP connection we have

$$\eta = \frac{E(\hat{Q})}{E(\tau)} \tag{1}$$

where $E(\hat{Q})$ and $E(\tau)$ denote the mean number of the corresponding quantities. Let there be n trains in a given cycle. Let θ_k denote the number of packets in the kth train given that we start with train 1 having one packet *i.e* $\theta_1 = 1$. Further, $\theta_k = \theta_{k-1} + 1$ in the congestion avoidance phase and $\theta_k = 2\theta_{k-1}$ during the slow-start phase. Let ϕ_k denote the round trip time taken by a packet of the kth train. A packet of the kth train can encounter a queue at the buffer. Let q_k be the queue

size at the gateway that the first packet of the kth train has to encounter. Further let μ denote the bottleneck bandwidth and σ denote the loss window size during the previous cycle. Hence, we have

$$\begin{aligned} E(\hat{Q}) &= E[E(Q/n,\sigma)] + E(\alpha) && (2)\\ &= \sum_{\sigma} P(\sigma) \sum_{n} P(n/\sigma) E(Q/n,\sigma) + E(\alpha) && (3)\\ E(Q/n,\sigma) &= \sum_{k=1}^{n} \theta_k && (4)\\ \tau &= \sum_{k=1}^{n+1} (\phi_k + q_k/\mu) && (5) \end{aligned}$$

where α are the number of packets sent by the source after the packet loss but before the packet loss is detected by the sender while Q is the number of packets sent by the source before the first packet loss. We assume that $E(\alpha) = E(\sigma)$. Let $E(\phi_k) = T$ where T is the average round trip time. A problem that arises at this point is how to determine the average time spent in the buffer during a cycle since it depends on the buffer occupancy. The buffer occupancy depends on the number of flows as well as on the train size of each flow. We assume that the first packet of every train in the flow of interest to us experiences the same mean delay in the buffer. We further let this mean delay be clubbed in the measure of the average round trip time T. Thus with this, the unaccounted time in the buffer for the first packet of the first train equals the time to service this packet which is $1/\mu$. In the subsequent train the unaccounted time equals $3/\mu$. Thus, this keeps on increasing until the window size equals W_m. Hence, if the window grows to W_m, the total unaccounted time in the queue during a cycle is given as

$$\frac{(W_m)(W_m+1)}{2\mu}$$

For simplicity, we assume that in other cases also where the window does not grow to W_m, the total queuing delay in a cycle is half of that given above. A main motivation for this approximation is to keep the analysis simple. Thus, with this we have

$$E(\tau) = (E(E(n/\sigma)) + 1)\, T + \frac{(W_m)(W_m+1)}{4\mu} \quad (6)$$

We next try to evaluate the expressions $P(\sigma), P(n/\sigma)$and $E(Q/n,\sigma)$ calling them as the window probability calculation, train probability calculation and the packet count calculation respectively.

4.1.1 Window Probability Calculation : TCP Tahoe

Let W_ρ denote the loss window size in the ρth cycle. l_1 and l_2 are the window sizes such that until size l_1 no packets are dropped and from size l_1 to l_2 packets are dropped with a probability p_1. From l_2 to the maximum window size W_m, the packets are dropped more aggressively with probability p_2 such that $p_1 < p_2$. Now $\{W_\rho\}_\rho$ denotes the sequence of window sizes at which packets are dropped in successive cycles. It is obvious that $\{W_\rho\}_\rho$ forms a Markov chain. Hence, in order to determine $P(\sigma),\ \ \sigma = l_1, l_1+1, \cdots, W_m$ we seek to characterize the probability $P(W_{\rho+1} = y+k/W_\rho = \sigma)$ also denoted as $P(y+k/\sigma)$ where y equals $\lfloor \sigma/2 \rfloor$ and is thus the slow start threshold. Let $q_1 = 1-p_1$ and $q_2 = 1-p_2$. Also let the function $s(x_1, x_2)$ be given by

$$\begin{aligned} s(x_1,x_2) &= (x_2+1)x_1 + \frac{x_2(x_2+1)}{2} \quad \text{if } x_2 \geq 0 \\ &= 0 \quad \text{if } x_2 < 0 \end{aligned}$$

Depending on the relative values of y, l_1 and l_2 we have three different cases, $y \leq l_1$, $l_1 < y \leq l_2$ and $l_2 < y$. For lack of space we show the expression for the transition probability only for the first case. Expressions for the other two cases also follow similarly.

Case 1: $y \leq l_1$

$$P(j/\sigma) = \begin{cases} q_1^{s(l_1, j-l_1-1)} \left[1 - q_1^j\right] & l_1 \leq j \leq l_2 \\ q_1^{s(l_1,l_2-l_1)} q_2^{s(l_2+1, j-l_2-2)} \left[1-q_2^j\right] & l_2 < j < W_m \\ q_1^{s(l_1,l_2-l_1)} q_2^{s(l_2+1, W_m-l_2-2)} & j = W_m \end{cases}$$

We can now generate the stationary distribution $P(\sigma),\ \ \sigma = l_1, l_1+1, \cdots, W_m$ from the above transition probability matrix using any of the standard methods.

4.1.2 Train Probability Calculation: TCP Tahoe

We next need to determine the probability distribution of (n/σ), the number of trains in a cycle given σ, the loss window size during the previous cycle. Given σ, the minimum number of trains in

a cycle is denoted by m_1 and the maximum number of trains in the cycle is denoted by m_3. Further, after m_2 trains have occurred in the cycle the dropping probability is increased to p_2. The values m_1, m_2 and m_3 are given as

$$m_i = \begin{cases} \lceil log_2(y) \rceil + 1 + l_i - y & y \leq l_i \\ \lceil log_2(l_i) \rceil + 1 & l_i < y \end{cases} \quad i = 1, 2 \tag{7}$$

$$m_3 = W_m - \lfloor \frac{\sigma}{2} \rfloor + \lceil log_2 \lfloor \frac{\sigma}{2} \rfloor \rceil \tag{8}$$

Hence, we now have the probabilities of the different trains given as

$$\begin{aligned} P(n_1 = \hat{m}_1 + k/\sigma) &= q_1^{\sum_{i=\hat{m}_1}^{\hat{m}_1+k-1} \theta_i} (1 - q_1^{\theta_{\hat{m}_1+k}}) \quad 0 \leq k \leq \hat{m}_2 - \hat{m}_1 \\ P(n_1 = \hat{m}_2 + k/\sigma) &= q_1^{\sum_{i=\hat{m}_1}^{\hat{m}_2} \theta_i} q_2^{\sum_{i=\hat{m}_2+1}^{\hat{m}_2+k-1} \theta_i} (1 - q_2^{\theta_{\hat{m}_2+k}}) \\ & \qquad 0 < k < \hat{m}_3 - \hat{m}_2 \\ P(n_1 = \hat{m}_3/\sigma) &= q_1^{\sum_{i=\hat{m}_1}^{\hat{m}_2} \theta_i} q_2^{\sum_{i=\hat{m}_2+1}^{\hat{m}_3-1} \theta_i} \end{aligned}$$

Calculating $E(n)$ given the above probability distribution can then be done as

$$E(n) = E[E(n/\sigma)] \tag{9}$$

$$= \sum_{\sigma} P(\sigma) \sum_{j} j P(n = j/\sigma) \tag{10}$$

4.1.3 Packet Count Calculation : TCP Tahoe

We next seek to characterize $E(Q/n, \sigma)$. The number of packets in a cycle depends on both n, the number of trains in the cycle and σ, the window size at the loss instant during the previous cycle. Once n and σ are given it is easy to calculate the number of packets in the cycle $E(Q/n, \sigma)$ which we proceed to do next.

$$E(Q/n, \sigma) = \begin{cases} \frac{(y+b)(y+b+1)}{2} - \frac{(y-2)(y-1)}{2} & n \geq log_2(y) \\ 2^n - 1 & log_2(l_1) \leq n < log_2(y) \end{cases} \tag{11}$$

where $b = n - log_2(y)$. At this point we have all the ingredients necessary to calculate the expected throughput in a cycle using equations 3 and 6.

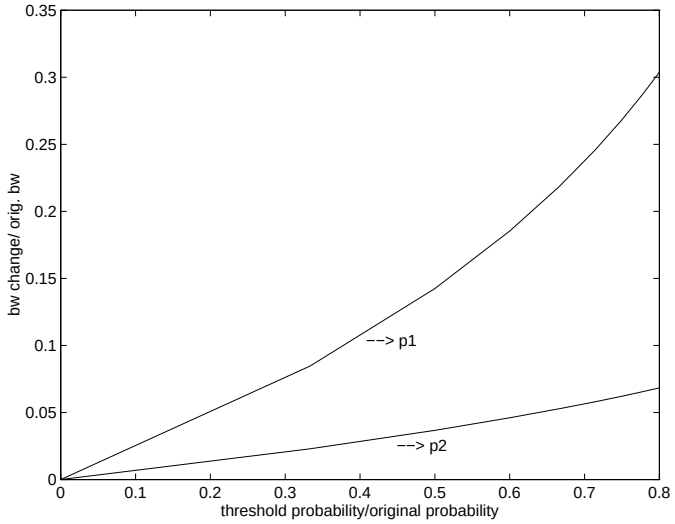

Figure 2: *Variation of flow bandwidth as a function of dropping probability variation for an adaptive flow*

Thus we now have an analytical model using which we next study the sensitivity of an adaptive flow to the different BRED parameters to determine which of these is most important to an adaptive flow. We can consider the five parameters namely p_1, p_2, l_1, l_2 and W_m as five control variables using which to control the bandwidth attained by a flow. In figures 2 and 3 we plot the change in the bandwidth attained by the TCP flow as a function of the change in a control variable while keeping the other four control variables constant. The control variable which is varied is marked against the curve shown. The change is calculated with respect to the first point corresponding to the best value of the control variable considered, for e.g. the lowest value of the drop probability or the highest value of the drop thresholds. As can be seen from figures 2 and 3, the throughput of a flow is more sensitive to p_1 as compared to p_2. This is understandable since large values of p_1 ensures that the window never grows to large values. Similarly of the three flow thresholds, the throughput is most sensitive to W_m and the least sensitive to l_2. This implies that for the adaptive flows the higher the value of W_m the higher the throughput attained. Further, this also implies that l_1 have a high value and thereby l_1 be as close to l_2 as possible.

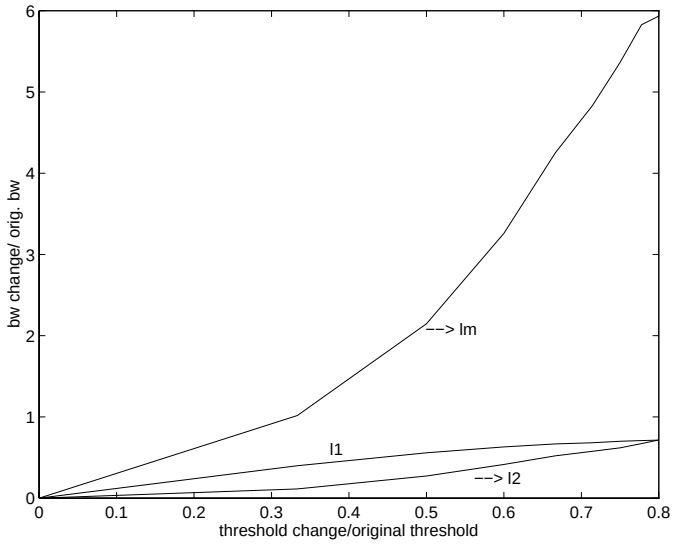

Figure 3: *Variation of flow bandwidth as a function of flow threshold variation for an adaptive flow showing sensitivity of the flow to the different flow threshold parameters*

4.2 Single Congested Node –CBR Flow

Since a cbr flow has no feedback any packet drops have no effect on the source rate. Thus the thresholds of l_1 and l_2 have no effect on cbr assuming the cbr input rate is high enough compared to the backbone bandwidth. Also with a packet drop probability of p, the cbr rate would be scaled down by $1-p$ assuming no buffer overflows. Thus, it is obvious that a cbr flow is sensitive to only W_m and p_2. Purpose of p_1 is to give an early warning which has no effect in case of cbr flows. Hence, in the algorithm BRED, based on the sensitivity of both adaptive and non-adaptive flows to the different parameters, we let p_1 to be far smaller than p_2 and l_1 to be close to l_2 with the exact difference being made clearer later. From the point of view of adaptive flows we also desire to have W_m large enough compared to l_2 but as noted above and as we also see later, this would confer a tremendous advantage on a non-adaptive flow.

4.3 Single Congested Node–Many Sources

4.3.1 TCP flows over BRED

While in the earlier analysis, we have analyzed the behavior of a tcp flow assuming an infinite buffer size, in practice it is not so. Consider a finite buffer size B. Assume N flows passing through a gateway. Since, these flows also share the pipe, we assume as earlier that the window size of a flow equals the buffer occupancy at the gateway. In such a case let W_m be the maximum window size allowed at the gateway after which all the packets of the flow start being dropped. Then a necessary and sufficient condition to prevent any buffer overflow is that

$$B > W_m N$$

For ease of analysis without loss of generality, in the sequel we let

$$\begin{aligned} l_1 &= \beta l_2 \quad 0 < \beta \leq 1 \\ W_m &= \alpha l_2 \quad \alpha > 1 \end{aligned} \tag{12}$$

Hence, in order to prevent buffer overflow we require

$$\alpha < \frac{B}{N l_2} \tag{13}$$

It is intuitive that the window size allowed to a flow in the absence of information about the round trip times should depend on the buffer size as well as the number of flows sharing the gateway. Hence, we assume that $l_2 = B/2N$ unless otherwise stated. The coefficient 2 in introduced in the denominator since we desire to keep some buffer space to accommodate the many short lived flows that characterize the internet. With this, we require $\alpha < 2$.

4.3.2 CBR flows over BRED

We next consider the performance of CBR flows passing through a BRED gateway. A misbehaving cbr flow tries to occupy as many buffer spaces as possible. Hence the buffer occupancy of each

non-conforming cbr flow is αl_2. In contrast each conforming flow can have atmost l_2 packets in the buffer on the average. The percentage buffer occupancy of the non-conforming flow then equals the throughput percentage obtained by it. Hence, a heuristic upper bound on the throughput of a non-conforming cbr flow can be obtained by assuming that it occupies all the possible buffer space while the conforming flows occupy $0.5\beta l_2$ buffer space on the average. Note that as the buffer space becomes smaller and smaller, this assumption reflects reality and hence in such a scenario the non-conforming flow is limited by the upper bound given below. As the buffer size grows, the amount of average buffer space occupied by the conforming flows also grows and hence the non-conforming flows get lot lesser than the possible upper bound. A heuristic expression for the upper bound on the throughput of a non-conforming flow η_{cu} in terms of percentage of the link bandwidth is then given as

$$\eta_{cu} = \frac{100\alpha}{\alpha n_c + 0.5\beta n_t} \tag{14}$$

where n_c is the number of non-conforming cbr flows and n_t is the number of conforming cbr and tcp flows. We show in the next section on simulation study that this bound is adhered to in the different scenarios considered. Thus, it is obvious that in order to reduce η_{cu}, α should be low enough such that $1 < \alpha < 2$. Further, β should be high enough such that $0 < \beta < 1$. This also suits well the earlier result which required that for an adaptive flow l_1 be as large as possible. Hence, based on the heuristic arguments and also on numerous simulation experiments, we choose $\alpha = 1.3$ and $\beta = 0.9$ for the simulations. Note that making α and β very close to each other would militate against the early warning mechanism for adaptive flows. Choosing α as given also helps to accommodate the short lived flows in the buffer. We have also verified that the behavior of the algorithm is robust to variations of α and β around the chosen values.

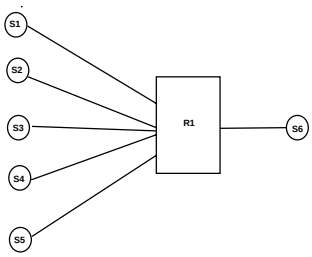

Figure 4: *Simulation Network*

5 Simulation Study

5.1 BRED Gateway Algorithm

In the previous sections we have described a gateway algorithm and analyzed it's behavior so as to obtain heuristic guidelines on setting the values of the different control parameters. Keeping this in mind, for our simulations the parameters are fixed as follows.

$$\begin{aligned} l_2 &= B/(2\lceil \hat{N} \rceil) \\ p_2 &= \sqrt{(\hat{N})}/(\sqrt{(\hat{N})} + 10) \\ p_1 &= p_2/10 \end{aligned} \tag{15}$$

where $\hat{N}$ is an estimate of the number of flows active at the gateway. It is calculated as

$$\hat{N} = (1 - w_n) * \hat{N} + w_n * Nactive \quad 0 < w_n < 1 \tag{16}$$

We have chosen $w_n = 0.02$.

The network topology that we used for the simulations consists of a single router with an outgoing backbone link as in [7] and [9]. The network is as shown in figure 4. The sources connect to the router through a feeder access link of 100Mbps. We experimented with different number of sources, though in the first set of graphs we show results with only 10 sources. The delay of the feeder links was varied for the different experiments. For the simulations, we assume that sources 4 and 5 and 10 are running non-adaptive applications while the rest of the sources are running adaptive applications. Adaptive sources are modelled by TCP source and sink nodes that implement a

congestion control algorithm equivalent to that of TCP tahoe. Further, the TCP flows denoted tcp1, tcp2, tcp3, tcp6, tcp7, tcp8 and tcp9 had rtts of .04s, .1s, .06s, .04s, 0.1s, 0.06s and 0.04s respectively excluding the service time which depends on the backbone link bandwidth. The packet sizes and window sizes for the TCP connections were constant during a single simulation but varied over the different simulations from sizes ranging from 100 to 500 bytes. Non-adaptive sources are modelled by constant bit-rate traffic. We use ns2 [10] for the purpose of simulations. Each simulation is run for 100s and the latter half of the simulation is used for the purpose of bandwidth calculation. Further, each experiment is repeated 5 times and the average taken over all the five trials. In the sequel all bandwidth values are in Mbps and hence the units may be omitted.

Figure 5a considers a backbone bandwidth of 1.5Mbps with 150 packets of buffer which is equivalent to 400ms of the backbone link speed. The cbr upper limit in this case as obtained from equation 14 is .2766 Mbps. Figure 5b considers a backbone bandwidth of 1.5Mbps with 50 packets of buffer. In these and subsequent figures, all tcp flows having the same rtt are marked using the same marker (either x, o or +). As can be seen irrespective of the sending rate of the cbr sources, the TCP flows are not shut out as in case of the other gateway algorithms like RED. This is achieved by keeping a little extra state. Further, with a reduced buffer size, the non-conforming flows are at an advantage and hence approach the upper bound as obtained from analysis. We have also verified through simulations the fact that if some of the non-adaptive flows have a rate within what is permissible at the gateway, then they are not penalized in any manner which is a very important requirement of a fair gateway algorithm.

We next study the performance of RED and BRED gateways for short-lived flows. The results are shown in figure6 with the cbr flows being denoted by squares and tcp flows with the same rtt being shown by the same marker. In these figures tcp1, tcp2 and tcp3 start at time 0 while tcp6, tcp7, tcp8 and tcp9 start at 30s. Among the non-adaptive flows, cbr4 starts at 10s, cbr5 at 20s and cbr10 at 50s and are assumed to have an input rate of 4.5Mbps each which is also the same as the rate of the backbone link. Further, tcp7, tcp8 and tcp9 end at 50s while cbr 10 ends at 70s. All the other flows are still active when the simulation ends at 100s. In this scenario tcp flows 7,

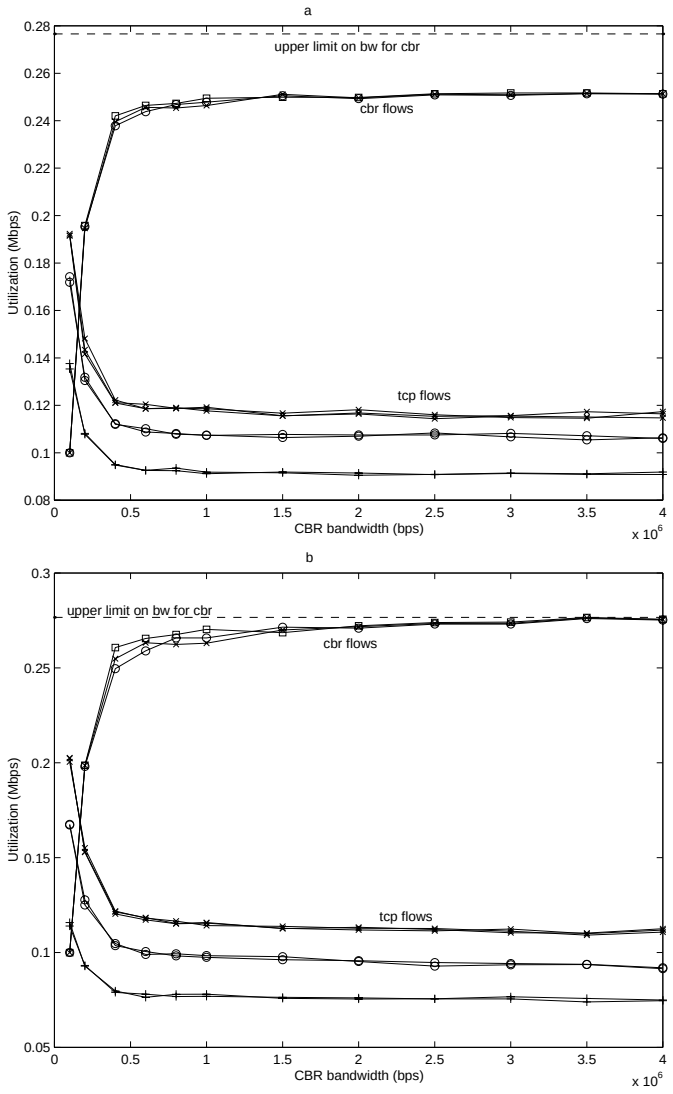

Figure 5: *Performance of BRED over 1.5Mbps link with 7 tcp and 3 cbr flows with Buffer size of a)150 packets and b) 50 packets*

8 and 9 model short lived tcp flows while cbr10 models a short lived cbr flow. It can be seen that in case of BRED gateways as seen from figure6b, the tcp flows are not shut out even when all the cbr flows are active and each sending at the rate of the backbone link. When 9 flows are active as between the time 30 and 50s, then the maximum possible rate for cbr from equation 14 is 1.0174 while after 70s when only 6 flows (2 cbr and 4 tcp) are active the upper limit on the cbr bandwidth from equation 14 is 1.3295. Between 50 and 70s when 3 cbr and 4 tcp flows are active, then the upper limit for the cbr flows is 1.0263. Thus it can be seen that even for short lived flows, BRED

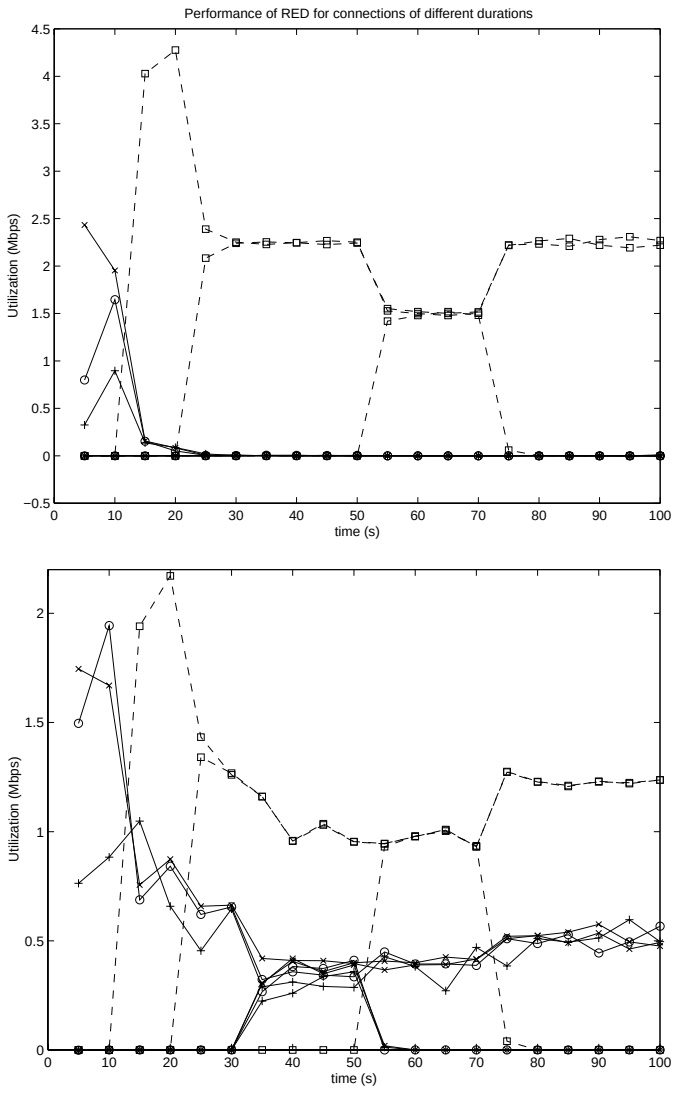

Figure 6: *Performance of a)RED and b)BRED gateways with short lived flows with a sampling interval of 5 sec and cbr input rate of 4.5Mbps each over a 4.5 Mbps link*

tries to ensure fairness in the division of the backbone bandwidth.

5.1.1 Effects of Buffer size

We now look at the effects of buffer size on the performance of BRED. We assume 10 flows as earlier with 3 of the flows running non-adaptive cbr traffic. The behavior is shown in figure 7. On the x-axis we plot the buffer size in terms of the backbone bandwidth seconds. The y-axis shows

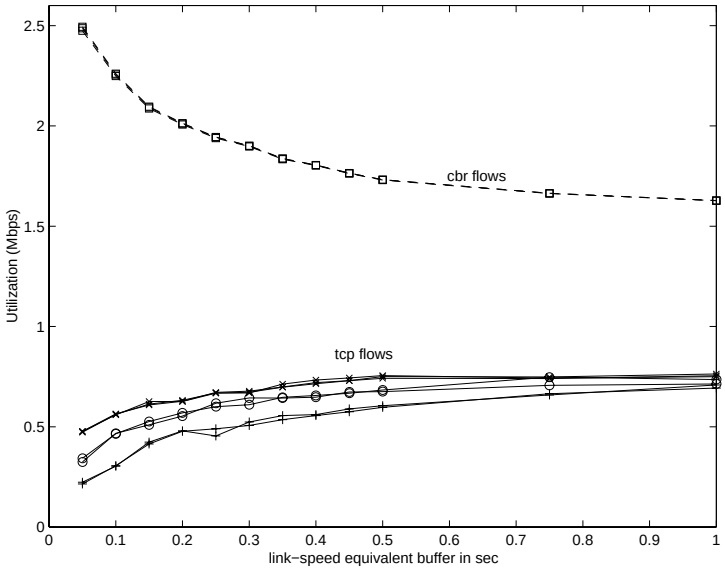

Figure 7: *Effect of Buffer size on performance of BRED with 10 flows and 10Mbps link*

the throughput of each of the 10 flows. In these simulations, each of the non-adaptive flows have an input rate equal to the backbone link bandwidth. As can be seen from figure 7, if the buffer size is very small, about 50 ms of the link speed, then the cbr flows do gain though the adaptive flows are not completely shut out as in RED even for this scenario. This is because of too many flows and a small buffer size due to which the effective buffer size for each flow is far less than its bandwidth-delay product. With a buffer size of 100 ms which is what is generally used in practice, the throughputs of the tcp flows increase much. Further increase in the buffer size though does not lead to very dramatic change in the throughput of the adaptive flows.

5.1.2 Fairness with many flows

In this subsection, we study the performance of BRED gateways in ensuring fairness amongst the different flows in the presence of many flows. The measure of fairness which we consider is the fairness quotient (F) [2] which is defined as

$$F = \frac{(\sum_{k=1}^{N} b_i)^2}{N \sum_{k=1}^{N} b_i^2} \tag{17}$$

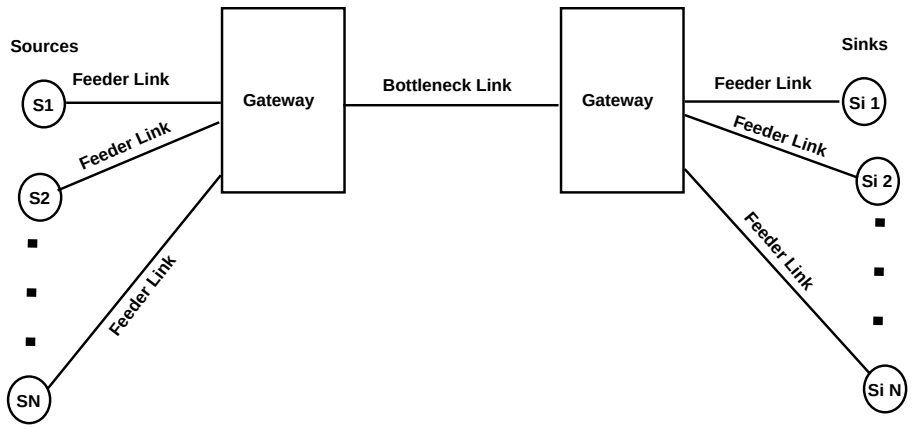

Figure 8: *Simulation Network for many flows*

where N is the number of flows and $b_i \;\; i = 1, \cdots, N$ is the bandwidth obtained by flow i. Some of the properties of this index [2] are that the fairness is bounded between 0 and 1 with a totally fair allocation having a fairness of 1 and a totally unfair allocation having a fairness of $1/N$. The fairness index is independent of scale and is also a continuous function.

We consider a network with a topology as in figure 8. The backbone link delay is assumed to be 6ms and the delay for all the feeder links to the sinks is assumed to be 4 ms with the bandwidth of each of these feeder links being 100Mb. For the feeder links from the source to the gateway, the bandwidth is 100Mb while the delay is uniformly distributed between 10 and 100ms. All the delay values are for the one-way delays. Thus we simulate flows with a rtt varying from 40ms to 220ms. One-fourth of the total number of flows in each case consist of non-adaptive cbr flows. The total bandwidth of all the cbr flows is changed over the different simulations and is shown on the x axis for all the figures. The fairness measure F, is plotted on the y axis. The system is simulated once for 100s and the bandwidth values taken over the latter part of the simulation. Unlike earlier, we do not consider averages 5 trials but consider just a single trial and take the values from this. The adaptive flows run TCP Tahoe. Packet sizes of all the flows are assumed to be 500 bytes as in many of the earlier cases. The receiver window is assumed to be large enough so as not to be a constraint during the tcp data transfer. Each source is assumed to have an infinite amount of data to send. BRED gateway algorithm is used over the backbone link while drop-tail algorithm is used

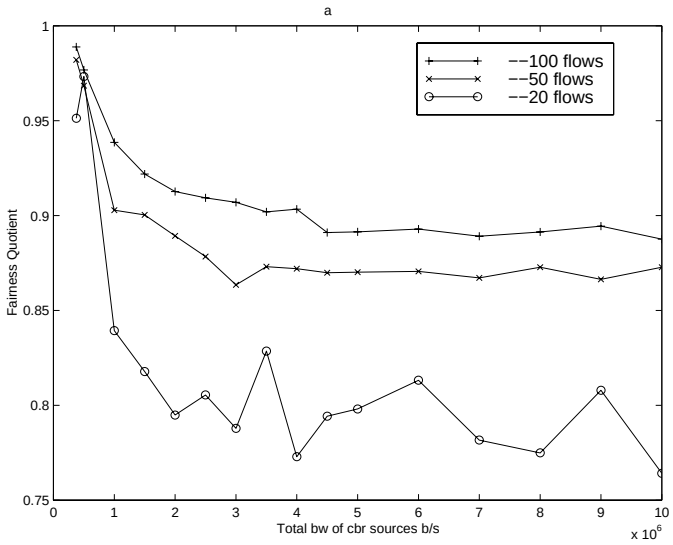

Figure 9: *Fairness of BRED with many flows over a backbone link of 1.5Mbps*

over the feeder links. This is okay since only one flow traverses any given feeder link. Further, the buffer size at the gateways in each simulation is assumed to be equivalent to 400 ms of backbone link bandwidth.

In figure 9 the backbone link bandwidth is 1.5 Mbps and we consider cases of 20, 50 and 100 flows. As can be seen from this figure, the performance of BRED is very good even with large number of flows and a high cbr input rate. In fact, as the number of flows increase, the performance of BRED gateway becomes more fair as expected with the fairness quotient values being about 90 percent. As the cbr flows increase their sending rate the fairness values decrease slowly till some point after which they taper off. Also note that with 20 flows, in figure 9, the fairness increases first and then decreases. This is because in the first trial the cbr sources are sending far less than their fair bandwidth share. But as they keep on increasing the input rates above their fair share, BRED algorithm ensures that the goodput that the cbr sources receive is limited.

5.2 Comparison with Other gateway Algorithms

5.2.1 Comparison with RED gateways-Adaptive flows only

We next compare the performance of BRED gateways with other gateway algorithms. While using RED gateways we let the lower threshold to be one-fourth of the buffer size and the upper threshold to be one-half of the buffer size. The maximum dropping probability is taken to be 0.1. It is accepted widely that RED ensures fairness while dealing with only adaptive flows. But as we show next, this is true only over a large time interval. BRED gateways on the other hand are very fair even over a short time interval. This is very much necessary in the internet which is characterized by many short and bursty flows.

We again use the simulation topology as shown in figure 4. 7 tcp flows are assumed to flow through the gateway with a backbone bandwidth of 0.8Mbps. The rtts of the tcp flows are as given previously. In figures 10a and 10b we compare the performance of RED and BRED gateways by making the bandwidth calculations every 2 seconds. In the graphs, curves with the same marker (e.g. + x or o) denote flows with the same rtt. As can be seen from these curves, BRED gateways ensure fair distribution of the bandwidths even over very short time intervals.

5.2.2 Adaptive and Non-adaptive flows

We next consider the case where both adaptive tcp flows and non-adaptive cbr flows share the gateway and look at the bandwidth distributions for the different flows considering a small time window of 5 seconds for a backbone link of 4.5Mbps. The cbr input rates are 4 Mbps each. The buffer size is 450 packets equivalent to 400ms of backbone link speed. The results are shown in figure 11. With RED, TCP flows are completely shut out by the cbr flows which split up the available bandwidth amongst themselves. In contrast we see that the performance is very much better in case of the BRED gateways.

Before concluding we also look at the fairness of LQD gateways. In order to facilitate comparison with BRED gateways, the fairness of LQD gateways over a 1.5Mbps link is simulated under the

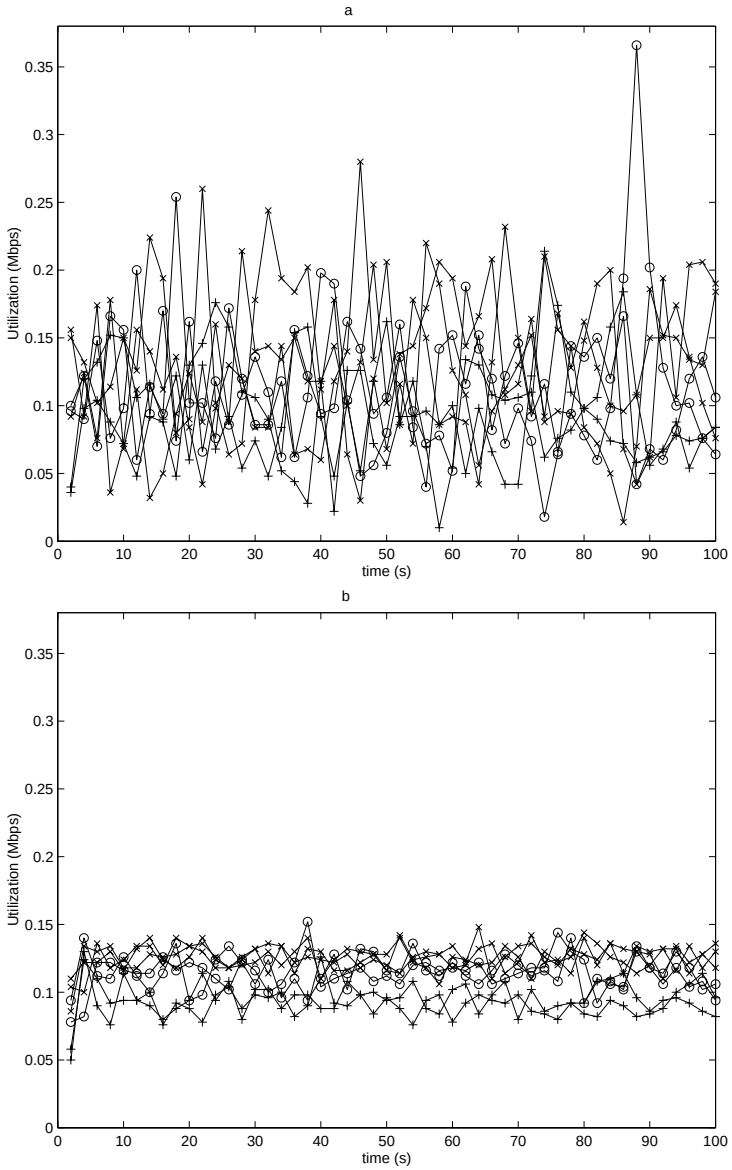

Figure 10: *Bandwidth distribution among 7 TCP flows with sampling interval of 2 sec with (a)RED and (b)BRED showing more time homogeneous allocation of bandwidth with BRED*

same conditions as used in figure 9. The results for the LQD gateway are shown in figure 12. Comparing this with figure 9 it can be concluded that not only are the LQD gateways difficult to implement but also the fairness quotient of the LQD gateways is far less than the fairness quotient of the BRED gateways.

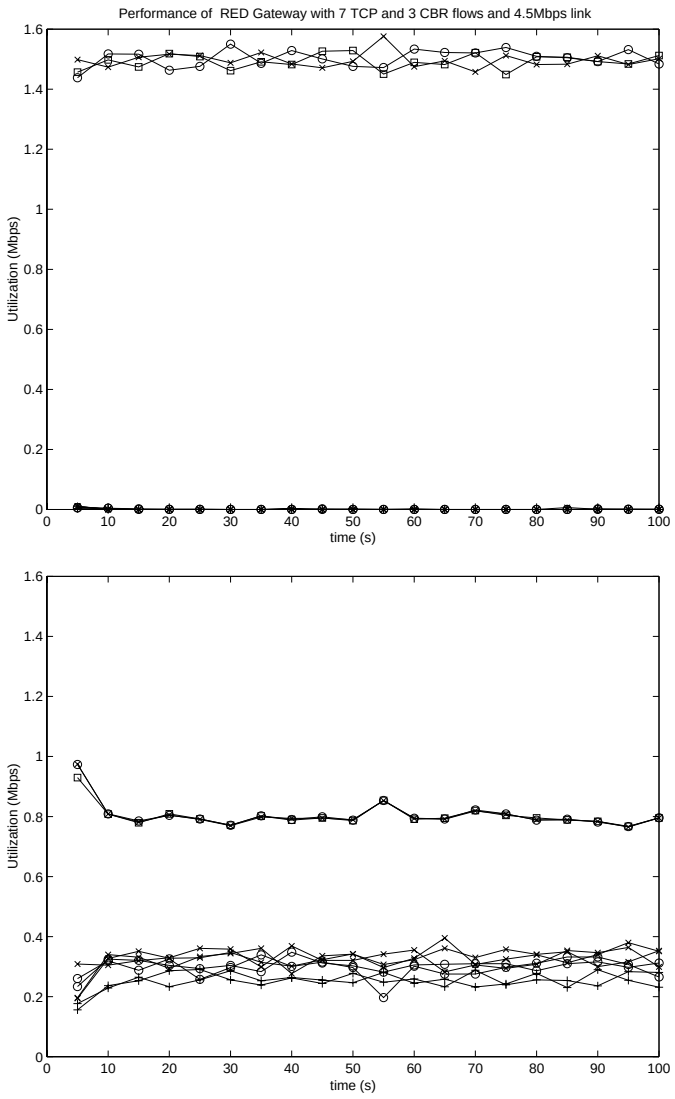

Figure 11: *Bandwidth distribution with sampling interval of 5 sec, 7 TCP and 3 CBR flows with a cbr input rate of 4Mbps each over a 4.5Mbps link showing BRED behavior over short time intervals*

6 Conclusion

In this paper we have proposed a gateway algorithm BRED which tries to ensure a fair division of the bandwidth of the link to which the gateway is attached, among the different flows sharing the gateway buffer. Not only is BRED simple to implement, but it is also scalable. Further, the fairness quotient of BRED gateways far exceeds the fairness quotient of the other gateway algorithms. Note that, in BRED we do not seek to explicitly identify the bad flows. We would also like to remark

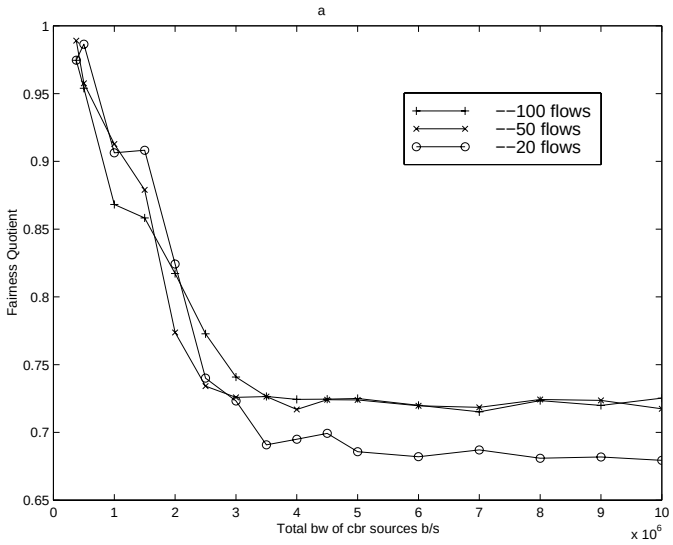

Figure 12: *Fairness with many flows over a 1.5Mbps backbone link with LQD gateways*

that BRED as given is addressed towards robust flows like Tahoe [12], New-Reno and Sack. While dealing with flows like Reno which are not robust [12], it will also be necessary to ensure that multiple packets are not dropped from a window. This may require another variable for each flow to track the number of successful packets after the dropped packet. The drop or accept decision may then need to be done based on both the queue occupancy and the number of packets accepted since the last packet dropped for each flow.

An approach which we plan to investigate in the future is to identify the bad flows on the basis of the buffer occupancy and thereby constrain the bad flows by having different set of flow limits on them. We do not do this in BRED so as to keep the algorithm very simple and also to ensure that the gateway has to do very little extra work. The approach of identifying bad flows and punishing them would require more work by the gateway and if this could be done then a better division of the bandwidth could be obtained. As part of the future study we also plan to look at BRED in the context of multiple gateways. Simulations with TCP reno are also planned as also with a mix of cbr flows, TCP Tahoe flows and TCP Reno flows. In this paper we have looked at the case of short bursty flows for small number of flows and shown that the BRED performance in such a case is also quite good. This study has also to be extended for the case of hundreds of short bursty flows coexisting with longer lived flows and for a more realistic sending pattern.

References

[1] Charny A. *An Algorithm for Rate Allocation in a Packet-Switching Network with Feedback,*. MIT/LCS/TR-601, Cambridge, 1994.

[2] D.Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN Systems*, 17(5), pp1-14 1989.

[3] Hashem E. *Analysis of random drop for gateway congestion control,*. MIT/LCS/TR-465, Cambridge, 1989.

[4] S. Floyd. Tcp and explicit congestion notification. *Computer Communication Review*, 24(5), Oct 1994.

[5] S. Floyd and K. Fall. Router Mechanisms to Support End-to-End Congestion Control, February 1997.

[6] S. Floyd and V. Jacobson. On traffic phase effects in packet switched gateways. *CCR*, April 1991.

[7] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Networking*, 1(4), August 1993.

[8] E. L. Hahne. Round-robin scheduling for max-min fairness in data networks. *IEEE Journal on Selected Areas in communications*, 9(7):1024–1039, September 1991.

[9] D. Lin and R. Morris. Dynamics of random early detection. In *Proc. ACM SIGComm Symp.*, Cannes, France, September 1997.

[10] UCL/LBNL/VINT Network Simulator. ns version 2. http://www-mash.cs.berkeley.edu/ns/.

[11] B. Suter, Lakshman T.V., D. Stiliadis, and A. Choudhury. Design considerations for supporting tcp with per-flow queueing. In *Proc. INFOCOM*, San Francisco, April 1998.

[12] Lakshman T.V. and U. Madhow. Performance analysis of window-based flow control using tcp/ip: the effect of high bandwidth-delay products and random loss. In *IFIP Transactions C-26, High Performance Networking*, pages 135–150, North-Holland, 1994.

[13] Lakshman T.V., A. Neidhardt, and T.J. Ott. The drop from front strategy in tcp over atm and its interworking with other control features. In *Proc. INFOCOM*, April 1996.