SRED: Stabilized RED

Teunis J. Ott
tjo@bellcore.com
Bellcore

T.V. Lakshman
lakshman@lucent.com
Bell Labs

Larry H. Wong
larryw@notes.cc.bellcore.com
Bellcore

Abstract—

This paper describes a mechanism we call "SRED" (Stabilized Random Early Drop). Like RED (Random Early Detection) SRED pre-emptively discards packets with a load-dependent probability when a buffer in a router in the Internet or an Intranet seems congested. SRED has an additional feature that over a wide range of load levels helps it stabilize its buffer occupation at a level independent of the number of active connections.

SRED does this by estimating the number of active connections or flows. This estimate is obtained without collecting or analyzing state information on individual flows. The same mechanism can be used to identify flows that may be misbehaving, i.e. are taking more than their fair share of bandwidth. Since the mechanism is statistical in nature, the next step must be to collect state information of the candidates for "misbehaving", and to analyze that information. We show that candidate flows thus identified indeed have a high posterior probability of taking a larger than average amount of bandwidth.

I. Introduction

Buffer management in routers can be made TCP aware so that both fairness and throughput are improved. A recent Internet draft (the "RED Manifesto", [1]) suggests using the Random Early Detection scheme proposed by Floyd and Jacobson [9]. This paper presents a number of ideas that can be used to strengthen RED-like mechanisms. Unlike for example in [10] the mechanisms described in this paper do not collect or analyze state information for individual flows.

The main idea is to compare, whenever a packet arrives at some buffer, the arriving packet with a randomly chosen packet that recently preceded it into the buffer. When the two packets are "of the same flow" we declare a "*hit*". The sequence of hits is used in two ways, and with two different objectives in mind:

- To estimate the number of active flows
- To find candidates for "misbehaving flow"

The definition of "hit" can be flexible. The strongest plausible requirement is to declare a "hit" only when the two packets indeed are of the same flow: same destination and source addresses, same destination and source port numbers, and same protocol identifiers. Alternatively, one could use a more lax definition of "hit", for example only same source address. We may also choose not to check for a hit for every arriving packet. Instead, we can test for hits for only a random or deterministic sub-sequence of arriving packets. Also, we can compare the arriving packet with not one, but with some $K > 1$ randomly chosen packets from the recent past. That would give information of the type "J out of K" hits which can be used to more accurately estimate the number of flows.

The ideas presented are applicable to ATM networks as well. In ATM, the virtual circuit identifiers are compared and a hit occurs when the compared cells are from the same virtual circuit. In that case it probably is wise not to declare a hit when the two cells are of the same AAL5 frame.

In this paper, we only consider IP networks. For simplicity, we assume fixed packet sizes and always choose $K = 1$ so that $J \in \{0, 1\}$. We do the comparison test for every arriving packet and declare a hit only when the two packets are from the same flow.

The paper is organized as follows: Section 2 describes a means of keeping information about flows that have recently sent packets. Section 3 describes how that data is used to define "hits", and how "hits" are used to estimate the number of active flows. Section 4 explains the importance of knowing the number of active flows, and describes a simple version of "Stabilized RED". Section 5 describes "Full SRED". Section 6 describes simulation results with all "persistent TCP" connections. Section 7 describes a more realistic source model and gives simulation results obtained with that more realistic model. Finally, Section 8 gives our conclusions.

II. Zombies

A simple way of comparing an arriving packet with a recent other packet is to compare it with a packet still in the buffer. This makes it impossible to compare packets more than one buffer drain time apart. To give the system longer memory, we augment the information in the buffer with a "*Zombie List*". We can think of this as a list of M recently seen flows, with the following extra information for each flow in the list: a "*Count*" and a "*time stamp*". Note that this zombie list or flow cache is small and maintaining this list is not the same as maintaining per-flow state. We call the flows in the zombie list "zombies".

The zombie list starts out empty. As packets arrive, as long as the list is not full, for every arriving packet the packet flow identifier (source address, destination address, etc.) is added to the list, the Count of that zombie is set to zero, and its timestamp is set to the arrival time of the packet.

Once the zombie list is full it works as follows: Whenever a packet arrives, it is compared with a randomly chosen zombie in the zombie list.

(1: Hit) If the arriving packet's flow matches the zombie we declare a "hit". In that case, the *Count* of the zombie is increased by one, and the timestamp is reset to the arrival time of the packet in the buffer.

(2: No Hit) If the two are not of the same flow, we declare a "no hit". In that case, with probability p the flow identifier of the packet is overwritten over the zombie chosen for comparison. The Count of the zombie is set to 0, and the timestamp is set to the arrival time at the buffer. With probability $1 - p$ there is no change to the zombie list.

Irrespective of whether there was a hit or not, the packet may be dropped if the buffer occupancy is such that the system is in random drop mode. The drop probability may depend on whether there was a hit or not.

In the simulation with which we tested out SRED, the timestamp is implemented but not further used. We anticipate that in

0-7803-5420-6/99/$10.00 (c) 1999 IEEE

a mature form of SRED the timestamp might be used, in case of a non-hit, to modify the probability that the incoming packet overwrites the zombie it is compared with: higher probability if the timestamp is older.

With the constant probability p that in case of a "non-hit" the incoming flow overwrites the zombie it is compared with, the time it takes for the zombie list to lose its memory is (roughly) M/p packets. In the choice of M and p, we conjecture that there is not much difference in performance between a sensible choice of M and p and an optimal choice. In this paper, we always use $M = 1000$, $p = .25$. The zombie list therefore loses its memory roughly once every 4000 packets. When there are only a few active sources, the system keeps its memory for a longer time because whenever there is a hit the zombie is not overwritten: instead more information is accumulated by increasing the $Count$.

If an arriving packet causes a hit, that fact by itself is evidence that the flow of the packet might be misbehaving, in the sense that "misbehaving" flows are more likely to cause hits than well behaved flows. If a packet causes a hit and the $Count$ of the zombie is high, the evidence becomes stronger. We also have the concept of "*Total Occurrence*" of a flow: this is the sum of $(Count + 1)$ over all zombies which have that flow. This is the total number of packets of the flow of which the zombie list has a record. We propose to compute this "*Total Occurrence*" of a flow only when a packet of that flow causes a hit with a high *Count*.

III. The Relationship Between Hits and Number of Active Flows

We maintain an estimate $P(t)$ for the hit frequency around the time of the arrival of the t-th packet at the buffer. For the t-th packet, let

$$Hit(t) = \begin{cases} 0 & \text{if } no\ hit, \\ 1 & \text{if } hit, \end{cases} \tag{3.1}$$

and let

$$P(t) = (1 - \alpha)P(t-1) + \alpha Hit(t), \tag{3.2}$$

with $0 < \alpha < 1$, for example (M and p as in section 2 above):

$$\alpha \sim \frac{p}{M}. \tag{3.3}$$

Then $P(t)$ is an estimate of the frequency of hits for approximately the most recent M/p packets before packet t. We can also consider this the probability that an arriving packet has a hit. In this paper we do not use (3.3) but $\alpha = \frac{1}{M} = .001$.

We now have the following.

Proposition 1: $P(t)^{-1}$ is a good estimate for the effective number of active flows in the time shortly before the arrival of packet t.

The argument to support the proposition is the following. Suppose there are many flows numbered $1, 2, \cdots$. Suppose that every time a packet arrives at the buffer, it belongs to flow i with probability π_i. We suppose these probabilities do not change over time. Hence, a zombie represents flow i with probability π_i. Then, for every arriving packet the probability that it causes a hit is

$$P\{Hit(t) = 1\} = \sum_i {\pi_i}^2. \tag{3.4}$$

When there are N active flows of identical traffic intensity: $\pi_i = \frac{1}{N}$ for $1 \le i \le N$, this gives

$$P\{Hit(t) = 1\} = \frac{1}{N}. \tag{3.5}$$

In this symmetrical case, Proposition 1 is exact or at least roughly unbiased. In general, we propose to use $P(t)^{-1}$ as an estimate for the *effective* number of active flows even in the asymmetrical case. For example, if we assume the random case with $\pi_i = 2^{-i}$ for $i \in \{1, 2, \cdots\}$ we have an expected value for $P(t)$ of

$$E[P(t)] = \frac{3}{16}. \tag{3.6}$$

Our estimate for the effective number of active flows is $\frac{16}{3} = 5.33$. From this we can roughly infer that flows taking more than $\frac{3}{16}$ of the total bandwidth are taking more than their fair share while those taking less than $\frac{3}{16}$ are taking less than their fair share. If utilization is high enough to warrant punitive action against bandwidth hogs, this action is taken against flows that use more than $\frac{3}{16}$ of the bandwidth. This will of course change the π's.

In general, when there are exactly N flows with probabilities $(\pi_i)_{i=1}^N$ (which sum to one) we have

$$\frac{1}{N} \le \sum_{i=1}^{N} \pi_i^2 \le 1. \tag{3.7}$$

The lower limit is achieved only when all π_i are the same, and hence are all equal to $\frac{1}{N}$. The upper limit is achieved only when one π_i is one and all other π_i-s are equal zero. The estimate of the effective number of active flows behaves in an intuitively acceptable way.

For the use proposed in this paper, the estimates need not be perfect. It is enough to have an approximate estimate for the number of active flows.

To reduce comparison overhead, it is allowable to update $P(t)$ not after every packet, but say after every L packets or at predetermined epochs. If we get H Hits out of L packets a possible update rule is

$$P(new) = (1 - L\alpha)P(old) + \alpha H. \tag{3.8}$$

As long as $0 \le L\alpha << 1$ this has practically the same effect as updating after every packet.

IV. Simple Stabilized RED

Unlike in RED [9], in our current scheme there is no computation of average queue length. The packet loss probability depends only on the instantaneous buffer occupation and on the estimated number of active flows. If after further investigation we find that adding computation of an average buffer occupation improves performance, that can easily be added. For the time being, it seems using an averaged buffer occupation does not improve performance.

In the remainder of this section we assume that all traffic entering the buffer is TCP. TCP's windowing scheme is an important factor in setting the drop probabilities.

This assumption that all flows are TCP indeed makes the number of active flows a very important entity. For example, suppose we have a certain queuelength caused by many, say 500, flows each with a small window of say 2 packets. Dropping one packet affects only one of the 500 flows. Assuming all flows were in congestion avoidance, all other flows increase their congestion window by one in the next round trip time. On the other hand, if there were only one flow with a window of 1000 packets, dropping one packet would reduce that window to 500, and it would take 500 round trip times for the window to grow back to 1000 packets.

Let the target buffer occupation be Q_0. Suppose we want to set a drop probability p, independent of the buffer occupation, which pushes the system to the target buffer occupation. Clearly, the optimal value of p depends on the number of active flows N. From the example above, p must be larger when N is large, and must be smaller when N is small, i.e. p must be an increasing function of N. This can be quantified as below.

It is known that if the drop probability is p and all N active flows are TCP flows moving "very large" files, every flow will get a congestion window ($cwnd$) of the order of $p^{-\frac{1}{2}}$ MSSs (Maximum Segment Sizes), see [4]. The actual average depends on whether there are delayed acknowledgements or not. The proportionality factor in the approximation is either close to $\sqrt{2}$ (no delayed acknowledgements) or close to 1 (with delayed acknowledgements). We boldly re-phrase this as

$$cwnd \sim p^{-\frac{1}{2}}, \tag{4.1}$$

this is the so-called "square root law".

For this discussion, the proportionality coefficient is not important. With N flows, the sum of the N congestion windows will be of the order of $N \times p^{-\frac{1}{2}}$ (MSSs, if we assume all flows have the same MSS). If we disregard the packets and acknowledgements in transit somewhere else in the network, we obtain the result that $N \times p^{-\frac{1}{2}}$ and Q_0 must be of the same order of magnitude. Here, Q_0 must also be expressed in MSSs. Boldly requiring equality instead of "same order of magnitude" we get:

$$N \times p^{-\frac{1}{2}} = Q_0, \text{ or } p = (\frac{N}{Q_0})^2. \tag{4.2}$$

The final result is that p must be of the order of N^2. This holds only over a limited range of p. We find that if p exceed values in the range .09 to .15, the drop rate is so high that TCP flows start spending much their time in time-out, resulting in degraded performance.

From the above arguments, we propose the following candidate drop probability function (which we denote by p_{zap}).

We have a buffer of capacity B bytes. A function $p_{sred}(q)$ is defined as follows:

$$p_{sred}(q) = \begin{cases} p_{max} & \text{if } \frac{1}{3}B \le q < B, \\ \frac{1}{4} \times p_{max} & \text{if } \frac{1}{6}B \le q < \frac{1}{3}B, \\ 0 & \text{if } 0 \le q < \frac{1}{6}B. \end{cases} \tag{4.3}$$

p_{max} in (4.3) is a parameter which in this paper we always choose equal to .15. We have also tried the values .12 and .09 and found this had no significant impact on the result.

When packet t arrives at the buffer, we first update $P(t)$ as described in section 3. In what we call "Simple SRED", if at the arrival instant the buffer contains q bytes, we drop the packet with probability p_{zap} which equals

$$p_{zap} = p_{sred}(q) \times min(1, \frac{1}{(256 \times P(t))^2}). \tag{4.4}$$

Interesting features of (4.3), (4.4) are (i) that in (4.3) p_{sred} has only three possible levels ($0, p_{max}$, and $p_{max}/4$) and does not depend on q in a continuous way, and (ii) that p_{sred} depends only on q, the instantaneous buffer occupation, and not on past behavior of q. If further analysis indicates these are worthwhile complications they can be added.

The motivation for choosing (4.3) and (4.4) in this manner is explained below. First, note that if we fix the drop probability independent of current buffer occupancy and dependent only on the estimate of the number of active flows then the buffer occupancy can vary considerably for a variety of reasons not accounted for in the determination of drop probabilities. This can happen because of widely varying round-trip times, flows using different MSSs, transients caused by new flows before they reach the equilibrium given by (4.1), etc. Therefore, we need to make the drop probabilities depend on the buffer occupancy. This is the role of p_{sred} in (4.4). It ensures that the drop probability increases when the buffer occupancy increases, even when the estimate $P(t)$ remains the same.

The ratio 4 in (4.3) has the effect that the drop probability quadruples when the buffer occupancy increases, with constant $P(t)$, from below $B/3$ to above $B/3$. Quadrupling the drop probability has the long term effect of halving the congestion windows. By choosing the ratio 4 we achieved that TCP connections reach the new equilibrium after a single packet loss.

In (4.3) we see that as long as

$$\frac{1}{256} \le P(t) \le 1 \tag{4.5}$$

we use the dropping probability

$$p_{zap} = \frac{p_{sred}}{65,536} \times \frac{1}{(P(t))^2} \sim \frac{p_{sred}}{65,536} \times (\text{number of flows})^2. \tag{4.6}$$

When however

$$0 \le P(t) < \frac{1}{256} \tag{4.7}$$

we use

$$p_{zap} = p_{sred}. \tag{4.8}$$

The latter is for two reasons: First, if the drop probability becomes too large, TCP flows spend much or most of their time in time-out, so further increasing p_{zap} is not sensible. Secondly, when $P(t)$ becomes small (when hits are rare), estimating $P(t)$ becomes unreliable.

The numerical results in the subsequent sections use $p_{max} =$ .15. In fact, we found that values of p_{max} in the range .09 to .15 all give very similar results. We suspect we can go somewhat below .09 and still see no significant change. Higher values are not a good idea because they merely serve to drive, in the situation (4.8), too many flows into time-out. Much lower values allow in the situation (4.8) relatively large congestion windows.

0-7803-5420-6/99/$10.00 (c) 1999 IEEE

The choice of 256 in 4.4 is arbitrary and needs further study. In section 5, we show that further improvements to p_{zap} are possible.

The version of SRED described above is what we call "Simple SRED". It has the characteristic that the drop probability of a packet only depends on the instantaneous buffer occupation q and on the estimate $P(t)$. In the next version of SRED, which we call "Full SRED" (or shortly SRED), the drop probability also depends on whether the packet caused a hit.

V. SRED: Stabilized Random Early Drop

In the simple SRED, we use hits only to estimate the number of active flows which are then used to set the dropping probabilities. We can also use hits directly in the dropping probabilities. This is based on the idea that misbehaving flows are likely to generate more hits. This is because misbehaving flows by definition have more packet arrivals than other flows and so trigger more comparisons. Secondly, they are more likely to be present in the zombie list.

In "Full SRED" we modify the drop probability as follows:

$$p_{zap} = p_{sred} \times min(1, \frac{1}{(256 \times P(t))^2}) \times (1 + \frac{Hit(t)}{P(t)}). \quad (5.1)$$

If a fraction π_i of all packets are from flow i, 5.1 implies that for flow i the zap probability (4.4) is multiplied by

$$\left(1 + \frac{\pi_i}{\sum_j \pi_j^2}\right). \quad (5.2)$$

This increases the drop probability for overactive flows and can also reduce TCP's bias in favor of flows with short RTTs.

VI. Simulation results for SRED

In this section, we show by simulations that SRED as defined in the previous section is indeed successful in keeping the buffer occupancy close to a specific target and away from overflow or underflow.

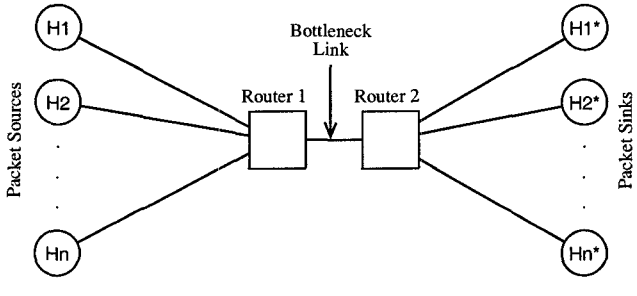

Fig. 1. Two Router Network

Figure 1 shows the network configuration used for the simulations. We have two routers connected through a DS3 link (45Mbit/sec) with a link propagation delay of 1 msec. Host H1 sends packets to Host H1*, H2 to H2*, etc. and destinations send back acknowledgement packets as required by TCP. The bottleneck link in Router 1 has a buffer of capacity .5 Mbytes. This translates to a buffer drain time of 88.8 msec.

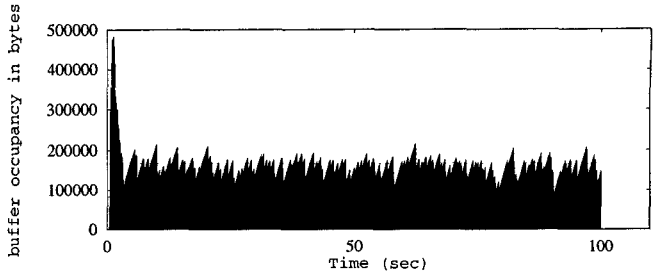

Fig. 2. SRED with 10 persistent connections, buffer occupation

The links between the hosts and routers are also DS3 links. We have sets of simulations called symmetrical runs and sets of simulations called asymmetrical runs. For symmetrical runs, all links between a host and a router have a propagation delay of 10 msec. In that case the minimal possible round trip time (RTT) is 42 msec. For the symmetrical runs, the propagation delay between Host H1 and its router, and between Host H1* and its router, is 1 msec, and for hosts with higher sequence number the propagation delays increase linearly to 20 msec. Thus, the minimal possible RTT varies linearly from 6 msec for pair H1 – H1* to 82 msec for the pair with the highest sequence number.

All flows have an MSS (Maximal Message Size) of 536 bytes. This is a typical segment size for TCP. TCP packets are usually 576 bytes long with 40 bytes being the typical header length. For the time being we assume persistent sources, i.e., a source will always send a packet whenever the congestion window permits transmission of a packet. Results for non-persistent sources are presented in a later section.

A. Symmetrical Network Simulations

Figures 2 – 8 show buffer occupancy behavior for what we call "Full SRED" with p_{max} = .15, for 10, 50, 100, 200, 300, 500, and 1000 persistent sources, on the symmetrical network. The buffer state is plotted using 10 msec samples.

All sources start at time zero a file transfer of an "infinitely large" file. All sources start in slowstart with the usual initial values for *cwnd* etc. The maximal window sizes (advertised windows) are large enough not to be a constraint. As result there is a traffic surge right after time zero. We have elected to show this surge, possibly at the cost of masking some of the finer detail later in the run. The versions of SRED and RED we use in our simulations do a good job (after the initial surge) of keeping the buffer occupancy below or at worst slightly above $B/3$, which translates in a delay of at worst slightly over 30 msec. One consequence is that loss is almost always due to the RED or SRED mechanism. Only in a few cases is there loss due to buffer overflow, and when this happens it is always during the initial surge.

The key point to note in the figures that when the number of flows is less than 256, SRED stabilizes the buffer occupancy to a level that is independent of the number of flows. This is what we expect since in this case 4.6 holds. When $N > 256$, where (4.8) applies and hence the dropping probability is independent of the number of active flows, the buffer occupancy increases

0-7803-5420-6/99/$10.00 (c) 1999 IEEE

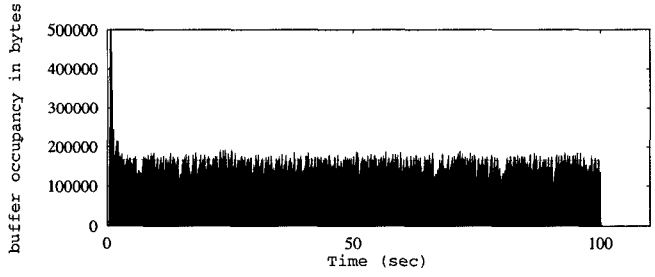

Fig. 3. SRED with 50 persistent connections, buffer occupation

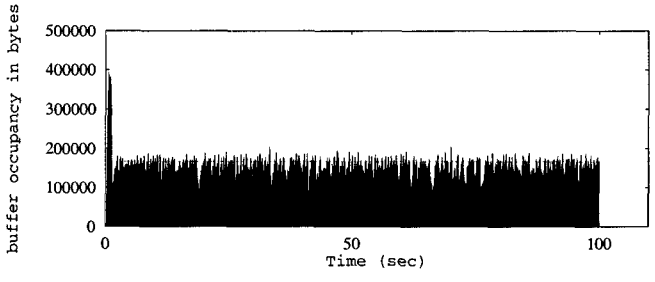

Fig. 4. SRED with 100 persistent connections, buffer occupation

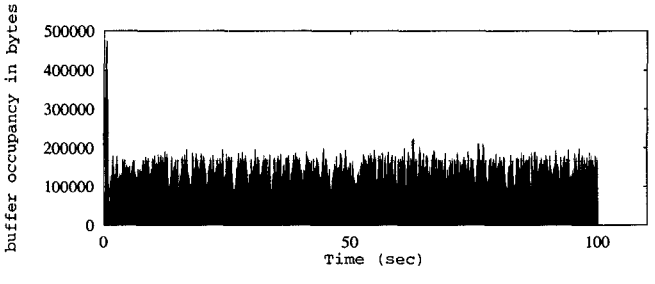

Fig. 5. SRED with 200 persistent connections, buffer occupation

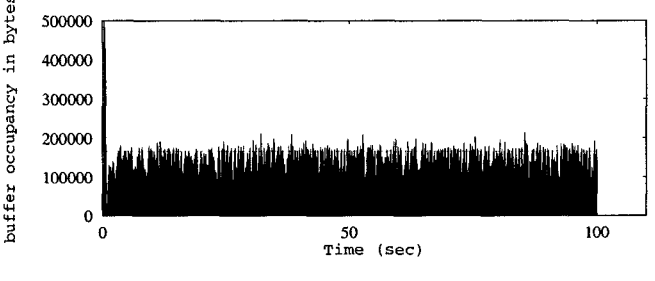

Fig. 6. SRED with 300 persistent connections, buffer occupation

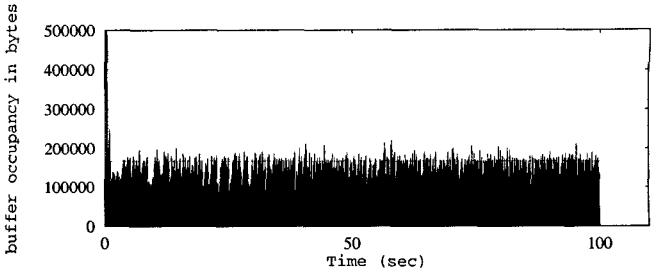

Fig. 7. SRED with 500 persistent connections, buffer occupation

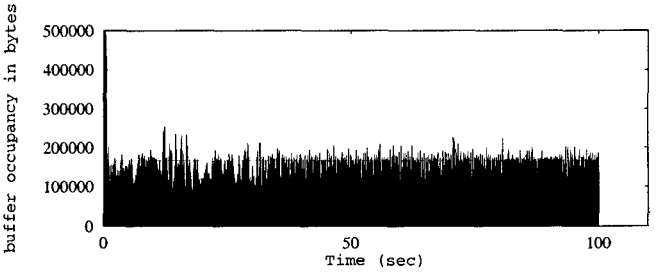

Fig. 8. SRED with 1000 persistent connections, buffer occupation

with N, although quite gradually so. (Due to the scale of the figures the increase is almost imperceptible). The reason for high numbers of flows we make the drop probability independent of that number of flows and let the buffer occupancy rise is that (as explained before) when the drop probability becomes high ($> .15$), too many time-outs happen, making TCP performance poor and highly random.

Another interesting observation is that the buffer occupation almost never decreases below $B/6$, i.e. to the point where the zapping probability becomes zero. This suggest we could make the band where $p_{sred}(q)$ equals $p_{max}/4$ even narrower, and put it closer to the empty level. We have not yet tried what will happen if we let this band disappear entirely. This may also be an interesting experiment. There is a risk involved in putting the lower boundary closer to the empty level: in the rare case of a very small number of active flows it could cause underflow while customers still want to send. The current set-up (no drop unless the buffer occupancy is 83.3 Kbytes or higher) does not run that risk.

Figures 9 – 11 show the values of $P(t)$ that were observed for the results in Figures 2 – 8. There is an initial transient period of 10 seconds when the values of P are high. This is because we start all TCP sources at the same time. They are all in the slow start phase and the initial traffic surge causes numerous time-outs (in particular in Figure 2). During that time there are much fewer than N active flows, so the high value of $P(t)$ is (almost) correct. Note that even during this time, the buffer occupancy is well stabilized (with the exception of Figure 2). After the transient period, the values of $P(t)$ decrease but remain somewhat high. For example, in the case of $N = 100$ sources, we observe that $P(t)$ fluctuates around .012 instead of around .01. This can be explained by (3.7): at any point in time not all 100 congestion

0-7803-5420-6/99/$10.00 (c) 1999 IEEE

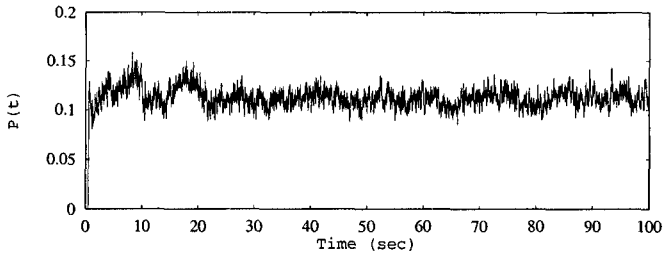

Fig. 9. SRED with 10 persistent connections and SRED, estimated P(t)

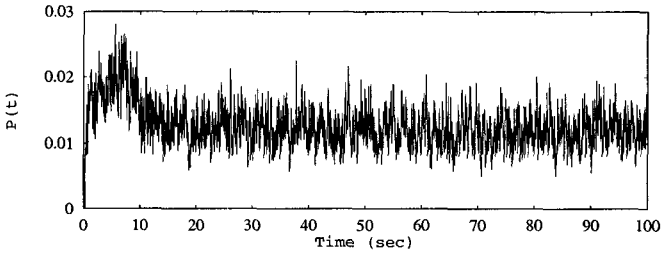

Fig. 10. SRED with 100 persistent connections, estimated P(t)

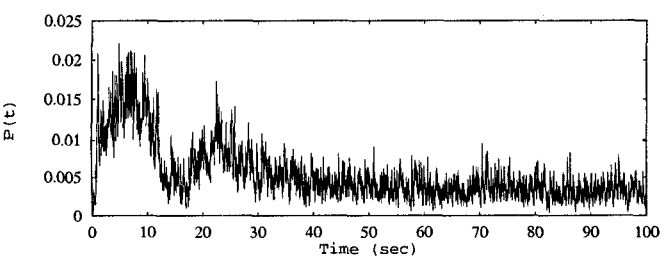

Fig. 11. SRED with 1000 persistent connections, estimated P(t)

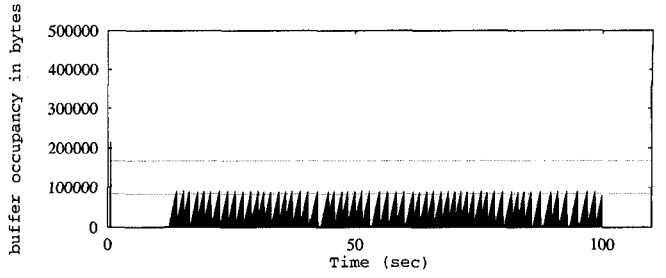

Fig. 12. RED with 10 persistent connections, buffer occupation

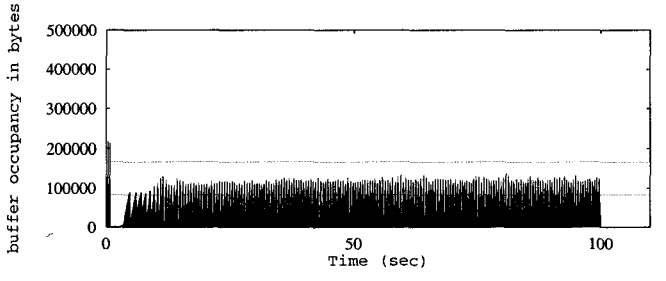

Fig. 13. RED with 100 persistent connections, buffer occupation

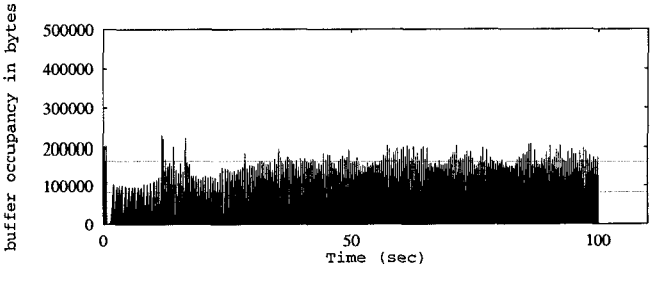

Fig. 14. RED with 1000 persistent connections, buffer occupation

windows are the same, so the estimated P comes out higher than $1/N = .01$.

Figures 12 – 14 show buffer occupation behavior in situations similar to those in Figures 2 – 8, but now with Random Early Detection (RED) as in [9]. This version of RED has "$w = 1/500$" and "$p_{max} = .075$" (see [9] for definitions of "w" and "p_{max}", note that "p_{max}" in this paper and in [9] have different definitions, the ratio of .5 makes them similar). The drop probability increases linearly from zero at $q = B/6$ to p_{max} at $q = B/3$. The comparison between RED and SRED is far from systematic. For a systematic comparison it would be necessary to compare optimized versions of the two.

We see that while for SRED the buffer occupancy is independent of the number of connections from 10 to 300 connections, and the buffer occupancy increases only slightly if the number of connections increases to 1000, for RED the buffer occupancy increases with the number of connections. Possibly more interesting, for SRED the buffer occupancy almost always is at least $B/3 = 83$ Kbytes. This suggests that the bands in (4.3) can be shifted downward without causing buffer underflow. This might also decrease the size of the initial burst of traffic right after time zero. For RED it seems undesirable to shift the bands down, see e.g. Figure 12. This is a likely area of further research.

Another interesting phenomenon is the slow recovery, after the initial burst at time zero, of RED with only 10 sources, and the "sawtooth" like behavior later on.

B. Asymmetrical Network Simulations

Figures 15 – 16 give individual throughputs of the 100 connections in the case of an asymmetrical network with 100 persistent TCP sources, for "Simple SRED" and "Full SRED".

As expected, we see that in the asymmetrical network connections with short round trip times get higher throughput. We also see that "Full SRED" reduces this advantage by a small but noticeable amount (compared with "Simple SRED"). The squared coefficient of variation of the 100 throughputs is .1039

0-7803-5420-6/99/$10.00 (c) 1999 IEEE

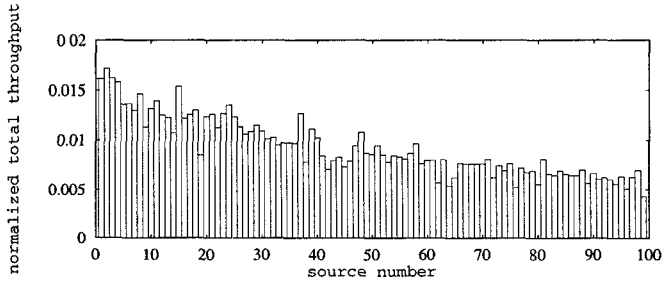

Fig. 15. 100 Asymmetrical Connections and Simple SRED, individual throughputs

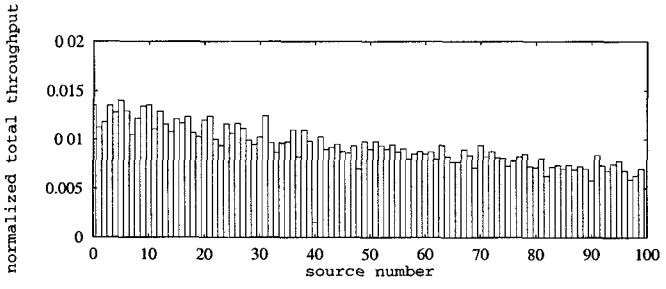

Fig. 16. 100 Asymmetrical Connections with Full SRED, individual throughputs

in the case of Simple SRED and .0522 in the case of Full SRED. This ability of Full SRED to counteract the impact of differential Round Trip Times increases when the number of active connections decreases. Routers that "never" carry a single flow that uses a significant fraction of the bottleneck bandwidth may have no use for Full SRED. Simple SRED and Full SRED are equally effective in stabilizing buffer occupation.

VII. A MORE REALISTIC SOURCE MODEL

In this section, we present a more realistic source model with random file sizes instead of infinitely long ones. Simulation results using these source models will be presented in the next section.

The distribution for the file sizes was derived by Neidhardt [2] from the Bestavros and Crovella measurements [5], [6] of web access by students at Boston University. The Neidhardt model has the the following distribution for the file sizes (in bytes).

$$P\{F > f\} = (1 + (\frac{f}{\mu})^{\alpha})^{-1}, \text{ where} \qquad (7.1)$$

μ is the median,

$$\alpha > 1,$$

$$\mu \times \alpha^{-1} \times \Gamma(\frac{1}{\alpha}) \times \Gamma(1 - \frac{1}{\alpha}) \text{ is the mean.}$$

If $0 < \alpha \leq 1$ (7.1) still is a probability distribution, but that distribution has no first moment.

For the Bestavros and Crovella data, Neidhardt found

$$\mu = 2190,\ \alpha = 1.15066\,.$$

Hence, the distribution (7.1) has a finite mean but infinite variance. The mean is 14954 bytes.

For the think time between transfers we used the distribution

$$P\{T > t\} = p_h(1 + (\frac{t}{\mu_h})^{\alpha_h})^{-1} + p_l(1 + (\frac{t}{\mu_l})^{\alpha_l})^{-1}, \qquad (7.2)$$

where

$p_h + p_l = 1$, i.e. a mixture of two distributions as in 7.1, and

$$p_h = 0.4953916,\ p_l = .5046084$$

$$\mu_h = 1.0,\ \alpha_h = 1.243437,$$

$$\mu_l = 0.0245032,\ \alpha_l = 3.252665.$$

The mean of this distribution is 2.18 seconds. However, after sampling, if the think–time came out below 50 msec we resampled. The resulting actual mean think time was about 3.1 seconds. The think time distribution found by Neidhardt from the Bestavros and Crovella data gives think times 10 times as large as those in (7.2). We divided the think times by 10 in order to get a significant load on the bottleneck link with number of connections in the order of a few thousand.

In our source model, the think time starts only after the the last byte of the previous file transfer has been acknowledged.

In our simulation, all thinktimes and filesizes are assumed independent. In the Bestavros and Crovella data, this is definitely not the case. We do not think this is material for the investigation at hand. In our simulation, when a new file transfer is about to start *ssthresh* is first reset to 64KBytes, *cwnd* is reset to 1 MSS, the transfer starts in *slowstart*, and possibly most important, the estimated RTT is reset to 3 sec, with an estimated standard deviation of 0 sec. This mimicks the opening of a new TCP connection as in http 1.0 (also in the case of an http 1.1 server with an http 1.0 client).

Figure 17 shows buffer occupancy behavior over time with 2000 TCP connections, each of which has source behavior as above. Figure 18 gives for the same same simulation the number of actually active sources as function of time. A source is considered "actually active" as long as it is sending a file, and it remains "actually active" until the last byte of the file has been acknowledged. We see that of the 2000 sources typically between 200 and 400 are active. This is an overestimate of the number of flows actually sending packets because while a source is waiting for its last acknowledgement it is not sending any packets and also connections in the early stage of slow start are not very active (the number of packets transferred per Round Trip Time is low). Figure 19 gives for the same simulation $P(t)$ as function of time. By comparing Figures 18 and 19 we see that $P(t)^{-1}$ underestimates the number of supposedly active flows. This is due to two causes: (1) many flows supposed to be active are actually not active yet, or not active anymore (as explained above) and (2) the active connections have varying congestion windows. Figure 20 gives the fraction of all active sources that

0-7803-5420-6/99/$10.00 (c) 1999 IEEE

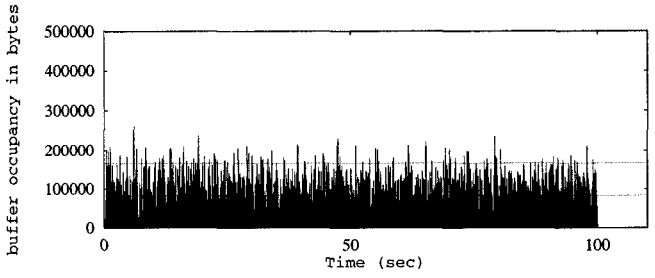

Fig. 17. SRED with 2000 "real" sources, buffer behavior

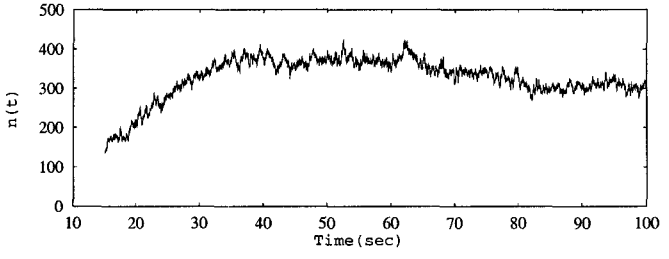

Fig. 18. SRED with 2000 "real" sources, number of active flows

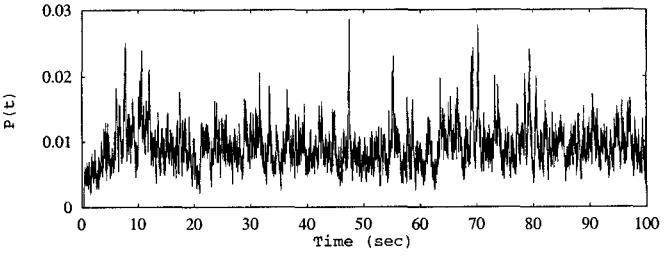

Fig. 19. SRED with 2000 "real" sources, P(t)

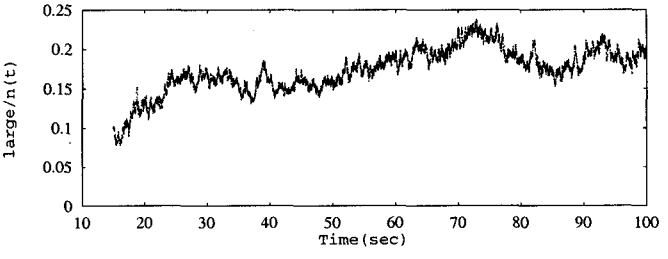

Fig. 20. SRED with 2000 "real" sources, fraction of flows that is large

are transporting a "large" file. A file is considered "large" if it contains more than 100 KBytes. Figure 21 is a plot of "filesize" (in bytes) versus "time" (time from sending the first byte until the last byte is acknowledged) of all files that were transported in runs already partially described in Figures 17 – 20.

Because (see Figure 17) the drop probability is often close to $p_{max}/4 = .0375$ and drop probability as high as $p_{max} = .15$ is not rare, files often lose packets. If this happens to be one of the first few packets, where the Round Trip Time estimation and the Standard Deviation estimation still have their original values, a long delay results. This explains the seemingly large number of files with transfer times of over 5 seconds. Of the total of 38,824 files for which the transfer times are given in Figure 21, 1,331 have a transfer time of over 5 seconds and 37,493 have a transfer time of below 5 seconds. The ones with transfer times below 5 seconds are presumably the files that do not lose any of their first 7 (or so) packets.

The combination of a quickly varying number of active flows, which transport mostly small files, and where if the file is large enough the *cwnd* tends to overshoot (4.1) during slowstart, explains the relatively wild behavior of the buffer occupation in Figure 17.

Figure 22 give the buffer occupation behavior for the same network as the Figures 17 –21, also with 2000 connections and the same source model, for RED.

Figures 23 – 24 give the buffer occupation behaviors on the same network with 3500 connections, for Full SRED and RED.

By comparing Figures 17 and 22 – 24 we see again that for SRED the buffer occupancy remains bounded away from empty, so that probably the bands in (4.3) can be shifted downward without causing buffer underflow. For RED it does not seems likely that the bands can be shifted downward without causing

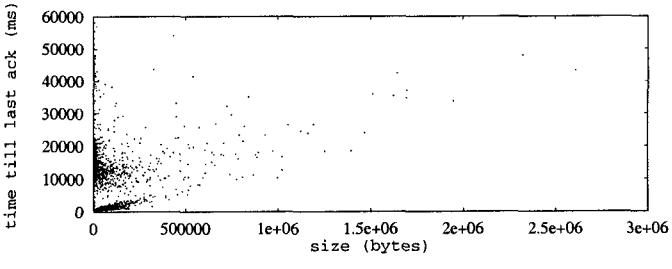

Fig. 21. SRED with 2000 "real" sources, durations

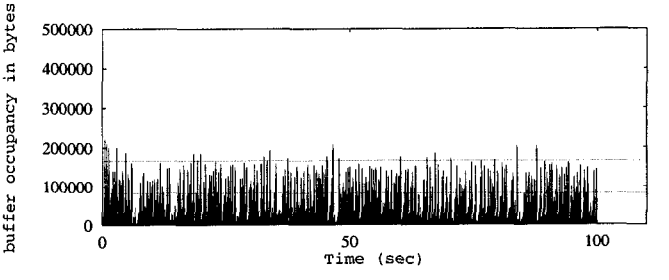

Fig. 22. RED with 2000 "real" sources, buffer occupation

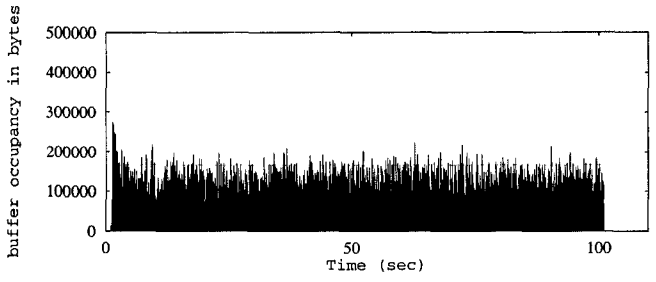

Fig. 23. SRED with 3500 "real" sources, buffer occupation

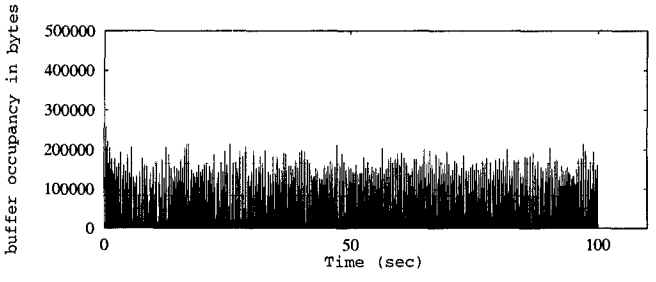

Fig. 24. RED with 3500 "real" sources, buffer occupation

buffer underflow. More work is needed in optimizing both RED and SRED, and then comparing optimized versions of these schemes.

VIII. Misbehaving Flows

As pointed out before, the hit mechanism can be used to identify candidates for misbehaving flows. This is because hits are more likely to occur for misbehaving flows. To gauge the ability of the hit mechanism and the zombie list to identify misbehaving flows, we simulated 100 persistent TCP connections and one "misbehaving" UDP connection. The UDP source sends a packet exactly once every 4 msec. Both TCP and UDP packets are 576 bytes long (this includes headers). The simulation was done for 100 seconds but the first 15 seconds are not used for statistics collection. Table 1 gives the total load and numbers of hits for the one UDP connection and the 100 TCP connections. For the UDP connection "load" is the number of packets it received during the last 85 seconds. For the TCP connections "load" is the sum of the increases in the "next expected" byte at the 100 destinations, divided by 536 to get MSSs. Doubtlessly some hits are on packets that were dropped after being tested for "hit".

	UDP Source	TCP Sources
Total Load in packets	20,821	808,300
Load/Connection	20,821	8,083
Total number of hits	494	11,000
Hits/packet	.0237	.0136

TABLE I
Load and Hits

	UDP Source	TCP Sources
Hits with Count = 0	452	10,484
Average *Total Occurrence*	27.9	17
Hits with Count = 1	38	495
Average *Total Occurrence*	28.2	18.1
Hits with Count = 2	3	20
Average *Total Occurrence*	27	18.9
Hits with Count = 3	1	1
Average *Total Occurrence*	29	12

TABLE II
Hits and Counts

Table 1 shows that the "overactive" connection, with a load of about 2.58 times that of the average "other" connection, has a hit/packet ration about 1.74 as large. This roughly agrees with the ratio of (π_i/π_j) theory predicts. As in (3.5) we see that more active flows not only send more packets that can cause hits, but also have a greater hit probability per packet. The hit probabilities for the UDP source are slightly smaller than what is predicted. This is because our UDP source sends packets at a perfectly constant rate whereas the TCP sources' rates fluctuate with the congestion windows.

An even stronger indication that a flow is taking more than its fair share is a hit with a high *Count* for the zombie. In addition, when a hit occurs we can compute the *total occurrence* in the zombie list of the flow which causes the hit. This *total occurrence* is the sum over all zombies that have the same flow of $(Count+1)$. Table 2 gives some more information about Hits and the *Counts* and average *Total Occurrences* they encounter.

For the "overactive" connection, a fraction .085 of all hits has $Count \geq 1$. For the TCP connection this fraction is .047. The overactive UDP connection also has higher average *Total Occurrences*.

Thus, a hit indicates a higher posterior probability that the flow in question is misbehaving. A hit with a high count increases the probability, and a hit with a high count and a high *Total Occurrence* increases the probability even further. These mechanisms can be used to filter flows and find a small subset of flows that are possibly misbehaving. These flows can then be monitored to determine if they are indeed misbehaving (such as by measuring and policing their rates, and possibly by comparing their actual rates with their contracted rates).

IX. Concluding Remarks

We presented a mechanism for statistically estimating the number of active flows in a bottleneck link. Our mechanism is based on the idea of comparing the flow identifier of incoming packets to those from a randomly chosen zombie in a zombie list that records information regarding flows that have recently sent packets to the link. A hit is said to occur when the comparison is successful. The number of active flows can be estimated from the average hit rate. The method does not require keeping per-flow state.

Next, we observe that for TCP flows the impact of a packet drop (in terms of a decreased arrival rate) is very high when

the bottleneck buffer occupancy is dominated by a few active flows with large windows and is very little when the bottleneck buffer occupancy is caused by a large number of connections with small windows. Hence, mechanisms like RED which try to control buffer occupancy can benefit by adjusting their drop probabilities using estimates of the number of active connections. We present schemes for adjusting drop probabilities such that the buffer occupancy is controlled to hover around a preset value independent of the number of active connections. Simulations using persistent as well as web-like traffic are used to show that the scheme is effective.

The hit mechanism can also be used to identify misbehaving flows without keeping per-flow state. Misbehaving sources are likely to get higher numbers of hits since they send more packets (resulting in more comparisons) and have a higher probability per packet of a hit (they have more entries in the zombie list). We also introduce the concepts of the *Count* and the *Total Occurrence* of a hit and show that a flow sending larger numbers of packets has more hits with high *Count* and high *Total Occurrence*. We give simulation results which confirm these observations.

Once a small number of possibly misbehaving flows have been identified they can be monitored closely to determine if they are indeed misbehaving. For some of these flows, actual behavior can be compared with their service contract.

Research is needed on how to set parameters in SRED, for example p_{max} and p_{sred} (see sections 4 and 5) as function of the buffer occupation. The version of SRED we used in our simulations does not use averaging of the bufferlength and has a "p_{sred}" function which takes on only three values. Research is needed on whether it is worth complicating the current version of SRED. Also, research is needed on how to set *Count* levels and *Total Occurrence* levels that declare a flow to be a candidate for "misbehaving". Even more research is needed on what to do once a flow has been declared "misbehaving".

REFERENCES

[1] Braden, B. et al (1998) Recommendations on queue management and congestion avoidance in the internet, IETF RFC (Informational) 2309. April 1998.

[2] Neidhardt, Arnold L. (1996) Traffic Source Models for the Bestavros and Crovella data. Private Communication.

[3] Mathis, M., Semke, J. Mahdavi, J. and Ott, T.J. (1997) The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *Computer Communications Review 27* (3), pp 67 - 82 (July 1997).

[4] Ott, T.J., Kemperman, J.H.B., and Mathis, M. (1996) The Stationary Behavior of Idealized TCP Congestion Behavior. ftp://ftp.bellcore.com/pub/tjo/TCPwindow.ps

[5] Cunha, C., Bestavros, A. and Crovella, M. (1995) Characteristics of of WWW client-based traces, Boston University Tech Report BU-CS-95-010, Boston, MA. July 1995.

[6] Crovella, M. and Bestavros, A. (1996) Self Similarity in World Wide Web Traffic: Evidence and possible Causes. *Proceedings of SIGMETRICS 1996.*

[7] W. Stevens, *TCP/IP Illustrated*, volume 1, Addison-Wesley, Reading MA, 1994.

[8] G.R. Wright and W.R. Stevens, *TCP/IP Illustrated*, volume 2, Addison-Wesley, Reading MA, 1994.

[9] Sally Floyd and Van Jacobson (1993) Random early detection gateways for congestion avoidance, *IEEE/ACM Transactions on Networking*, August 1993.

[10] Lin, D. and Morris, R. (1997) Dynamics of Random early Detection *Proceedings of SIGCOMM'97*