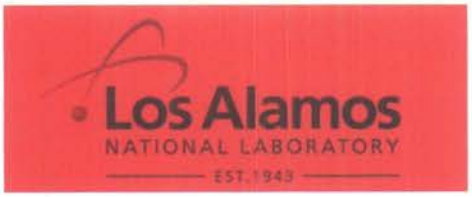
LA-UR- 07-2140

*Approved for public release;*
*distribution is unlimited.*

| | |
|---|---|
| *Title:* | LOW OVERHEAD ROUTER-BASED CONGESTION CONTROL TECHNIQUES TO PROTECT RESPONSIVE TRAFFIC |
| *Author(s):* | Venkatesh "NMI" Ramaswamy, Leticia "NMI" Cuellar, Stephan J. Eidenbenz, & Nicolas W. Hengartner, CCS-3 |
| *Intended for:* | IEEE 2007 GLOBECOM Conference<br>Washington, DC<br>November 26-30, 2007 |




Los Alamos National Laboratory, an affirmative action/equal opportunity employer, is operated by the Los Alamos National Security, LLC for the National Nuclear Security Administration of the U.S. Department of Energy under contract DE-AC52-06NA25396. By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes. Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy. Los Alamos National Laboratory strongly supports academic freedom and a researcher's right to publish; as an institution, however, the Laboratory does not endorse the viewpoint of a publication or guarantee its technical correctness.

# Low Overhead Router-Based Congestion Control Techniques to Protect Responsive Traffic

Venkatesh Ramaswamy, Leticia Cuéllar, Stephan Eidenbenz and Nicolas Hengartner
Computers and Computational Sciences
Los Alamos National Laboratory
Email: {vramaswa,leticia,eidenben,nickh}@lanl.gov

*Abstract*— **In this paper, we present queue management algorithms with low implementation complexity that can partition the bandwidth of an outgoing link among flows in a high speed packet switch. These algorithms belong to a family of queue management schemes called Sending Rate Estimate Based Queue Management Schemes (SREQM), which try to prevent congestion by effectively limiting flows based on their estimated sending rates. We also present techniques based on sampling and Bloom filters to further reduce the implementation overhead. The capability of the algorithms to protect responsive flows from non-responsive flows are confirmed by exhaustive analysis and simulations.**

## I. Introduction

The increasing interest in router-based queue management schemes that can enforce bandwidth guarantees to network flows is a consequence of the exponential growth of applications such as the packet video and packet audio, which do not implement congestion control [1]. This kind of traffic that do not implement congestion control are generally termed as *non-responsive traffic*. Non-responsive traffic can consume all the available bandwidth starving legitimate TCP flows (*responsive traffic*) in a network. In the past, when the number of flows traversing a router was in the order of a few thousands, the above problem was solved using complex scheduling schemes such as weighted fair queueing (WFQ) or its variants. With the present day routers carrying more than a million flows, implementing schemes such as WFQ is becoming painfully hard [2]. Therefore, researchers are moving towards more scalable solutions based on queue management schemes that can provide quality of service (QoS) guarantees to flows.

A comprehensive survey of router based queue management schemes that try to guarantee QoS to flows can be found in [3]. Most of these schemes cannot guarantee fairness among flows, while being inexpensively implementable at the router at high speeds. In [4], we develop a class of queue management schemes called sending rate estimate based queue management schemes (SREQM) that can guarantee max-min fair bandwidth allocation to flows in a router. These queue management schemes operate by differentially dropping packets from flows based on an estimate of the sending rate of flows as well as on the aggregate queue length. Simply stated, the probability of dropping packets from flow $j$ is a compound function of the estimate of the relative sending rate of flow $j$ and the aggregate queue length of the router buffer at time $t$. Because the amount of per-flow state needed is the bare minimum required to guarantee max-min fairness [5], the flow memory can be implemented using static random access memories (SRAM), making packet processing scalable with link speeds.

The queue management schemes that belong to the family of SREQM estimate the relative sending rate of flows using an exponentially weighted low-pass filter [3], $H_j(\cdot)$, for flow $j$ as

$$H_j(t) = \left(1 - \frac{1}{T}\right) H_j(t-1) + \frac{1}{T}\tilde{\chi}_j(t). \quad (1)$$

where $T$ is the time-constant [6] of the filter, and

$$\tilde{\chi}_j(t) = \begin{cases} 1 & \text{if the packet at time } t \text{ is from flow } j \\ 0 & \text{otherwise.} \end{cases}$$

The estimation works as follows. We assign a state variable $H_j(\cdot)$ for each flow, and initialize it to zero. When a packet arrives from a flow, the state-variable of the flow from which the packet arrived is updated as $H_j(t) = \left(1 - \frac{1}{T}\right) H_j(t-1) + \frac{1}{T}$, whereas the state-variable of all other flows are updated as $H_j(t) = \left(1 - \frac{1}{T}\right) H_j(t-1)$. The state-variable $H_j(\cdot)$ at any point of time is an estimate of the relative sending rate of flow $j$.

The objective of this work is to design algorithms to alleviate the processing overheads associate with the SREQM based schemes. In Section II, we develop an algorithm based on a modified estimator which does not require the update of all the state-variables on packet arrival. We then discuss some guidelines to set the parameters of the algorithm in Section III. A fluid model based algorithm along with some simulation results are presented next. We give two techniques based on sampling and Bloom filters to reduce the complexity of SREQM based algorithms in Section V. Finally, we conclude in Section VI

## II. Algorithms for Protecting Responsive Traffic and Bandwidth Partitioning

The Efficient SREQM (ESREQM) algorithm presented in [7], though capable of providing bandwidth guarantees to flows, requires us to update the state variable every time a packet arrives. This approach can be computationally very tedious. In this section, we present an algorithm called Low Overhead SREQM (LO-SREQM), to mitigate the computational overhead introduced by updating the state variable of all the flows on a packet arrival. This computational overhead

can be reduced at the cost of memory. Suppose that we assign an ID to all the packets that arrive at the queue, and a packet arrives every time-slot. The ID of the $i^{th}$ packet to arrive at the queue is assigned $i$, and therefore, the ID of the last packet will be the total number of all packets that arrived at the queue. If we can also keep track of the ID of the last packet that arrived from each flow, then we develop an alternate estimate of the relative sending rate of each flow that is computationally efficient. Define $\eta_j$ = ID of the last packet that arrived at the queue - ID of that last packet that arrived from flow $j$. Suppose at time $t$, a new packet from flow $j$ arrives. The new estimate of the relative sending rate of flow $j$, $\tilde{H}_j(\cdot)$, is given by

$$\tilde{H}_j(t) = \left(1 - \frac{1}{T}\right)^{\eta_j} \tilde{H}_j(t - \eta_j) + \frac{1}{T}. \quad (2)$$

With this procedure, we do not have to update the relative sending rate estimate of all the flows on every packet arrival. The new queue management scheme based on this modified estimate is described in Algorithm 1. When a packet arrives, we update the state variable of the flow from which the packet arrives based on the modified estimator given in (2). If the state-variable of that flow is more than the fair share parameter, $K$, the packet is dropped; otherwise the packet is admitted into the queue. The fair share parameter changes with the change in the number of users as well as changes in the network conditions. To accommodate these changes, we modify $K$ based on the level of congestion. If the aggregate queue size is more than some maximum threshold $q_{max}$, then we decrease the value of $K$. On the other hand, if the aggregate queue size is less than some minimum threshold $q_{min}$, then we increase the value of $K$. The rate at which we change $K$ has also an impact on the level of punishment we impose on flows that send at a rate higher than their fair share. This rate is determined by the parameter $F$. Some guideline on setting the parameters are discussed later in this paper.

We note in passing that, in many cases of practical importance, we need to treat flows differently. For example, we may want to allocate more bandwidth to a video stream than to a data transfer session. We can readily modify the estimator $\tilde{H}_j(\cdot)$ to treat flows differently. Associate a positive real number $w_i$, to flow $i$, representing its weight. We assign a weight to a flow, proportional to the amount of bandwidth we want to allocate to that flow. Denote the weighted estimator for flow $i$ to be $\mathcal{H}_i(\cdot)$. $\mathcal{H}_i(\cdot)$ can take a form as below

$$\mathcal{H}_i(t) = \frac{\frac{\tilde{H}_i(t)}{w_i}}{\sum_{j=1}^{n} \left(\frac{\tilde{H}_j(t)}{w_j}\right)}. \quad (3)$$

It is not difficult to establish that if we use $\mathcal{H}_i(\cdot)$, instead of $\tilde{H}_i(\cdot)$, in the algorithm, the bandwidth allocation to flow $i$ will be proportional to its weight $w_i$. This modified estimator can be used to treat flows differently in all other modifications of SREQM that we discuss in later sections of this chapter.

In the following theorem, we analyze the packet spacing required to minimize the packet drops in order to achieve maximum throughput.

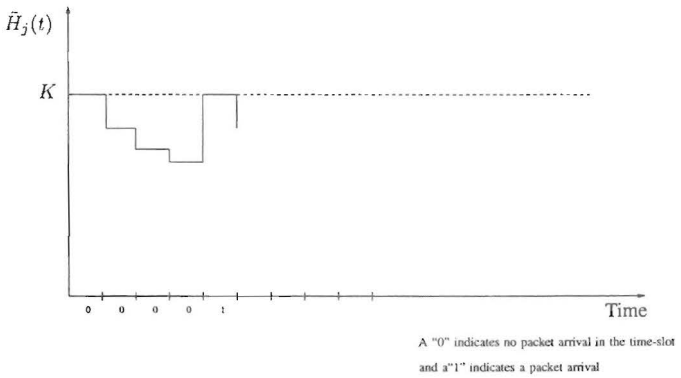



Fig. 1. Dynamics of $\tilde{H}_j(t)$

*Theorem 1: In LO-SREQM, in order to maximize the throughput, each source must equispace its packets. This spacing, m, in number of time slots, is given by*

$$m = \frac{\log(1 - \frac{1}{K})}{\log(1 - \frac{1}{T})} \quad (4)$$

*Proof:* Refer Figure 1. Let at $t = 0$, $\tilde{H}_j(0)$ be $K$. If the source $j$ transmits packet at the next time slot, $\tilde{H}_j(1)$ will exceed $K$, and that packet will be dropped. Therefore, in order to prevent any packet loss, the source must wait until $\tilde{H}_j(t)$ becomes $K - 1$. The number of time-slots required to reduce $\tilde{H}_j$ from $K$ to $K - 1$ can be computed as follows. Let the number of time-slots required to reduce $\tilde{H}_j$ from $K$ to $K - 1$ be $m$

$$\tilde{H}_j(0) = K \quad (5)$$
$$\tilde{H}_j(1) = (1 - \frac{1}{T})K \quad (6)$$
$$\tilde{H}_j(2) = (1 - \frac{1}{T})^2 K \quad (7)$$
$$\tilde{H}_j(m) = (1 - \frac{1}{T})^m K. \quad (8)$$

Since at the end of $m$ time-slots, the value of $\tilde{H}_j$ would have reduced from $K$ to $K - 1$, we have

$$\tilde{H}_j(m) = (1 - \frac{1}{T})^m K \quad (9)$$
$$= K - 1. \quad (10)$$

That is,

$$K - 1 = (1 - \frac{1}{T})^m K. \quad (11)$$

Solving the above equation, we get

$$m = \frac{\log(1 - \frac{1}{K})}{\log(1 - \frac{1}{T})}. \quad (12)$$

■

**Algorithm 1** LO-SREQM :: onPacketArrival(packet $P$)

```
PacketCount ⇐ packet id of P ;
x ⇐ flow id of packet P;
η_x ⇐ PacketCount − S[x];
S[x] ⇐ packet id of P;
H̃_x(t) ⇐ (1 − 1/T)^{η_x} H̃_x(t − η_x) + 1/T;
if (H̃_x(t) ≤ K/T) then
    add packet P to the queue;
else
    drop packet P;
end if
if (count > 0) then
    count−−;
else
    if (queue size < q_min) then
        K++;
        count ⇐ F;
    end if
    if (queue size > q_max); then
        K−−;
        count ⇐ F;
    end if
end if
```

## III. Setting the Parameters of LO-SREQM

In this section, we depend heavily on simulation results to determine the right set of parameters for our algorithm. There are three main parameters that we need to set: the time-constant of the filter $T$, the fair share parameter $K$, and the congestion parameter $F$. Setting the parameters to the right values is crucial for the proper functioning of the algorithm. Our goal is to achieve max-min fairness among flows, and in this section, we give some engineering guidelines to set the parameters of the algorithm so as to achieve max-min fairness.

### A. Time-constant $T$, and Fair Share Parameter $K$

Consider the case with three constant bit rate (CBR) flows with flow 1 sending at 100 Kbps, flow 2 sending at 200 Kbps and flow 3 sending at 300 Kbps. The relative sending rate of these flows are 0.167, 0.334 and 0.5, respectively. We study the performance of the estimation algorithm with different values of $T$. When $T = 30$, the variance in the estimation is large as shown in Figure 2, but the estimator approaches the relative sending rates almost immediately. On the other hand when $T = 20000$, the estimator has a very large bias and it takes around 600 sec to estimate the relative sending rate correctly, but with a very low variance. This is shown in Figure 3. The value of $T$ should be selected such that $T$ is not too big nor too small. Based on simulations, we recommend a value of 400 for $T$. For the above case the performance of the estimator with $T = 400$ is shown in Figure 4.

Regarding the fair share parameter, $K$, we observed that the initial value of the parameter $K$ does not have any major impact on the performance of the algorithm [3]. The value of $K$ is dynamically varied based on the level of congestion. The initial value of $K$ is set to 50 in all our simulations.

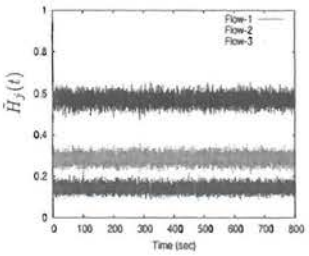



Fig. 2. Estimation of relative sending rates of flows when $T = 30$

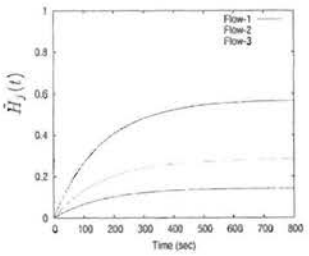



Fig. 3. Estimation of relative sending rates of flows when $T = 20000$

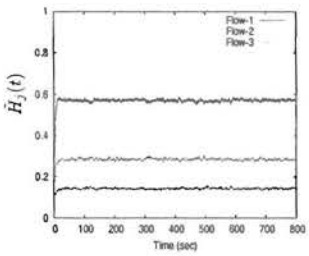



Fig. 4. Estimation of relative sending rates of flows when $T = 400$

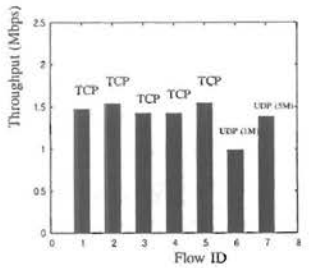



Fig. 5. Throughput received by seven flows on a 10Mbps link under the proposed algorithm.

### B. Congestion Parameter $F$

The congestion parameter $F$, has a profound effect on the performance of the algorithm and it is the most difficult parameter to set. $F$ determines how fast we change the value of the parameter $K$. To understand the effect of parameter $F$, consider the case where four TCP flows and a UDP flow are traversing a bottleneck dumbbell topology [2]. The capacity of the access links as well as the bottleneck link was set to 10Mbps. Since five flows are competing for 10Mbps bandwidth on the bottleneck, the fair sending rate of each flow would be 2Mbps. Suppose that $K$ is initially set to achieve a fair allocation. Since the UDP flow is sending at a much higher rate than its fair rate, all its packets will be dropped, making UDP throughput to zero. If we set $F$ too high, the value of $K$ will not be changed and the flows will not get a fair share. If we make $F$ too low, the UDP flow can achieve a higher average rate than its fair share, because the packet dropping phase completes rapidly as $K$ changes fast.

Table I shows the effect of the parameter $F$ on the average throughput of different flows. When $F$ is large, the UDP flow gets very low throughput; and when $F$ is low, the UDP flow gets a high share of the bandwidth. A value of approximately 200 for the parameter $F$ will result in a fair allocation and this is the value that we assign to $F$ in our simulations. We can also observe from the table that the scheme is not terribly sensitive to small changes in the value of the parameter $F$.

TABLE I

EFFECT OF THE CONGESTION PARAMETER $F$, ON THE THROUGHPUT OF DIFFERENT FLOWS

| Flow ID | Average Throughput (Mbps) | | | |
|---|---|---|---|---|
| | $F = 50$ | $F = 200$ | $F = 500$ | $F = 10000$ |
| 1 (TCP) | 1.922 | 1.933 | 2.105 | 2.502 |
| 2 (TCP) | 1.845 | 1.881 | 2.134 | 2.413 |
| 3 (TCP) | 1.886 | 1.853 | 2.111 | 2.335 |
| 4 (TCP) | 1.977 | 1.960 | 2.024 | 2.447 |
| 5 (UDP-5Mbps) | 2.376 | 2.080 | 1.178 | 0.593 |

There is no value of $F$ that will provide max-min fairness in all situations. If $F$ is small, some flows may get more bandwidth than their fair share. On the other hand, if $F$ is large, the flows that are sending at a higher rate than their fair share may receive lower bandwidth than their fair share. However, in both cases, the degradation of performance is gradual, as seen from Table I.

## IV. A FLUID FLOW MODEL ALGORITHM

In a fluid flow model, we assume that the number of packets generated by a flow is so large that it appears as a continuous flow of bits [8]. This assumption is particularly valid when the number of flows is large. In this section, we develop a simple algorithm, assuming the flows to be a continuous stream of bits, that can restrict all bottle-necked flows to their max-min fair rate.

Suppose that the outgoing link has a capacity of $C$ packets/sec. Assuming all flows to be equal, the fair share of each flow is $\frac{C}{n}$, where $n$ is the number of flows. Previously we showed that $\tilde{H}_i(\cdot)$ is an estimate of the relative sending rate of flow $i$. Therefore, if $\tilde{H}_i(\cdot) \leq \frac{1}{n}$, *i.e.,* the flow is sending at a rate less than its fair share, then the packets from that flow need not be dropped. On the other hand, if the flow is sending at a rate greater than its fair share, that is $\tilde{H}_i(\cdot) > \frac{1}{n}$, then a proportion of the packets equal to $\frac{\tilde{H}_i(\cdot)-\frac{1}{n}}{\tilde{H}_i(\cdot)}$, which forms the excess from the fair share should be dropped to bring the throughput of the flow to its fair share. This suggests that a simple probabilistic algorithm that drops a packet from a flow $j$ with a probability $P_d^j$ equal to

$$P_d^j = \max\left(0, 1 - \frac{1}{n\tilde{H}_j(\cdot)}\right) \tag{13}$$

can achieve fair bandwidth allocation and prevent malicious behavior. It is easy to see that the punishment a flow receives for sending at a rate higher than its fair share depends on $\frac{1}{n\tilde{H}_j(\cdot)}$. We can increase the severity of punishment by increasing the exponent of $\frac{1}{n\tilde{H}_j(\cdot)}$ to a positive integer, say $k$. The dropping probability will now take a form as follows

$$P_d^j = \max\left(0, 1 - \left(\frac{1}{n\tilde{H}_j(\cdot)}\right)^k\right). \tag{14}$$

The value of $k$ affect the penalty profile of the algorithm. The larger the value of $k$, greater the penalty imposed.

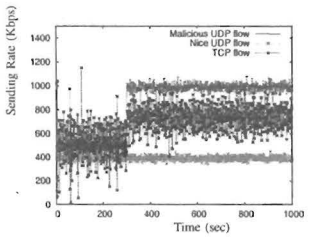



Fig. 6. Sending rate of three flows on a link with bandwidth 2Mbps under the proposed algorithm.

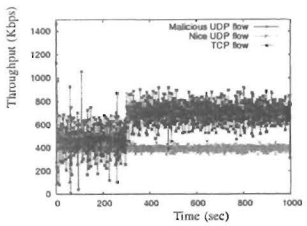



Fig. 7. Throughput achieved by three flows on a 2Mbps bottleneck link under the proposed algorithm.

For flows with different weights, the dropping probability should be modified as

$$P_d^j = \max\left(0, 1 - \frac{w_j}{\tilde{H}_j(\cdot)}\right), \tag{15}$$

where $w_j$ is the weight assigned to flow $j$. We term this new algorithm as Fluid SREQM, and is described in detail in Algorithm 2.

**Algorithm 2** Fluid SREQM :: onPacketArrival(packet $P$)

1: $PacketCount \Leftarrow$ packet id of $P$ ;
2: $x \Leftarrow$ flow id of packet $P$;
3: $\eta_x \Leftarrow PacketCount - S[x]$;
4: $S[x] \Leftarrow$ packet id of $P$;
5: $\tilde{H}_x(t) \Leftarrow \left(1 - \frac{1}{T}\right)^{\eta_x} \tilde{H}_x(t - \eta_x) + \frac{1}{T}$;
6: Mark packet $P$ for dropping with probability $\max\left(0, 1 - \frac{1}{n\tilde{H}_x}\right)$;
7: **if** (packet $P$ not marked) or (queue size $< q_{min}$) **then**
8: add packet $P$ to the queue;
9: **end if**
10: **if** (queue size $> q_{max}$) **then**
11: drop packet $P$;
12: **end if**

We now present some examples to show that Fluid SREQM works well in practice. In the first example consider three flows, of which one is a TCP flow and the other two are UDP flows, on a congested link with 2Mbps bandwidth. One of the UDP flows is a malicious flow that starts with a sending rate of 500Kbps, but increases its sending rate to 1000Kbps after 300s. The second UDP flow is a legitimate UDP flow sending constantly at a rate of 400Kbps. Figure 6 plots the sending rate of the three flows, while Figure 7 shows their throughput. Clearly, the malicious UDP is restricted to its fair share.

In the second example, we consider 32 flows on a dumbbell topology with a bottleneck link of 10Mbps. Flows indexed one to 30 are TCP flows, while the flows indexed 31 and 32 are UDP flows. The sending rate of flow 31 is 1Mbps and that of flow 32 is 5 Mbps, which means that both these flows are sending at a rate higher than their fair rate. Table II show the throughput received by each of the 32 flows. From the table it is clear that the algorithm can allocate bandwidth fairly among flows.

TABLE II

AVERAGE THROUGHPUT (BPS) RECEIVED BY 32 FLOWS ON A 10MBPS LINK.

| Flow ID | Average Throughput | Flow ID | Average Throughput |
|---|---|---|---|
| 1 (TCP) | 310177 | 17 (TCP) | 309089 |
| 2 (TCP) | 300273 | 18 (TCP) | 301697 |
| 3 (TCP) | 301169 | 19 (TCP) | 307105 |
| 4 (TCP) | 306289 | 20 (TCP) | 306993 |
| 5 (TCP) | 302977 | 21 (TCP) | 311825 |
| 6 (TCP) | 299809 | 22 (TCP) | 293313 |
| 7 (TCP) | 306033 | 23 (TCP) | 305281 |
| 8 (TCP) | 304913 | 24 (TCP) | 299201 |
| 9 (TCP) | 305905 | 25 (TCP) | 300273 |
| 10 (TCP) | 302481 | 26 (TCP) | 305457 |
| 11 (TCP) | 304513 | 27 (TCP) | 301905 |
| 12 (TCP) | 305681 | 28 (TCP) | 306049 |
| 13 (TCP) | 306609 | 29 (TCP) | 303745 |
| 14 (TCP) | 308337 | 30 (TCP) | 306801 |
| 15 (TCP) | 303697 | 31 (UDP-1Mbps) | 298936 |
| 16 (TCP) | 302449 | 32 (UDP-5Mbps) | 303952 |

In the third example, we consider seven flows on a bottleneck link of bandwidth 10Mbps. Of the seven flows, flows indexed six and seven are UDP flows sending at 1Mbps and 5Mbps, respectively. All other flows are TCP flows. Figure 5 shows the box plot of the throughput received by each flow. Again, Fluid SREQM successfully limits flows with very high relative sending rates and fairly allocates bandwidth to all the flows.

## V. METHODS TO REDUCE COMPLEXITY

In the following sub-sections, we present two techniques, based on sampling and Bloom filters, to further alleviate processing overhead and memory requirements associated with the algorithms presented in this paper. These techniques come at the expense of reduced accuracy in the relative sending rate estimation procedure, and therefore, may be inappropriate for applications that require accurate data at small time-scales.

### A. Sampling

In all the previously discussed algorithms, we update the state variable $\tilde{H}_j(\cdot)$ for each on packet arrival. For high speed links, which carry millions of flows, updating the state-variable on every packet arrival may be computationally prohibitive. One straightforward approach to mitigate this complexity is to update the state-variable with a probability $p$ on each packet arrival. For example, if there are $10^6$ flows, and we update $\tilde{H}_j(\cdot)$ once in every thousand packet arrivals, the number of updates we save is 999. Represent the modified estimate as $H_j^p(\cdot)$. Since $H_j^p(\cdot)$ attempts to approximate $\tilde{H}_j(\cdot)$ using a fewer number of updates, errors are unavoidable. These errors introduced by sampling the packets will depend on the nature of incoming traffic. If the incoming traffic is constant bit rate (CBR) traffic, then we only need very few samples to estimate its rate; on the other hand, if the sending rate of the incoming traffic is varying rapidly, we need more samples to estimate the sending rate with bounded errors. We can compute the expectation and variance of the modified estimate $H_j^p(\cdot)$, when the incoming traffic is Poisson with arrival rate $\lambda_j$ as

$$\mathbf{E}\left[H_j^p(t)\right] = \frac{\lambda_j}{\sum_{i=1}^n \lambda_i}\left(1 - e^{\left(-\frac{\sum_{i=1}^n \lambda_i}{T} pt\right)}\right), \quad (16)$$

and

$$\mathbf{Var}\left[H_j^p(t)\right] = \frac{\lambda_j}{\sum_{i=1}^n \lambda_i}\left(\frac{1}{2T-1}\right)\left[1 - \frac{\lambda_j}{\sum_{i=1}^n \lambda_i}\right] + \left(\frac{\lambda_j}{\sum_{i=1}^n \lambda_i}\right)^2 e^{-2\frac{\sum_{i=1}^n \lambda_i}{T} pt}. \quad (17)$$

From the above equations, we can see that the steady state values are

$$\mathbf{E}\left[H_j^p(t)\right] = \frac{\lambda_j}{\sum_{i=1}^n \lambda_i}, \quad (18)$$

and

$$\mathbf{Var}\left[H_j^p(t)\right] = \frac{\lambda_j}{\sum_{i=1}^n \lambda_i}\left(\frac{1}{2T-1}\right)\left[1 - \frac{\lambda_j}{\sum_{i=1}^n \lambda_i}\right] \quad (19)$$

which correspond to the steady state values of $\mathbf{E}\left[\tilde{H}_j(\cdot)\right]$ and $\mathbf{Var}\left[\tilde{H}_j(\cdot)\right]$ [3]. For Poisson traffic, as long as $p > 0$, at steady state, sampling does not have an effect on performance of the estimator. However, sampling does affect the time it takes for the estimator to reach the steady state.

The above results may not hold for traffic such as the TCP traffic, as they vary at a much higher rate. For TCP traffic we need to sample more frequently to accurately estimate the relative sending rate. In short, if we have some information about the nature of the incoming traffic, then we can decide on a sampling procedure, which can approximate $\tilde{H}_j(\cdot)$ with a given level of accuracy.

### B. Bloom Filters

In both the schemes described in this paper, we need to maintain a state-variable for each flow. For high speed links where there are millions of short flows, maintaining a state-variable for each of the flows, may be cumbersome, though possible. We can reduce the amount of bookkeeping by mapping many flows to a single state-variable. In this section, we present a technique, based on Bloom filters [9], to reduce the complexity of SREQM based queueing schemes at a reduced accuracy.

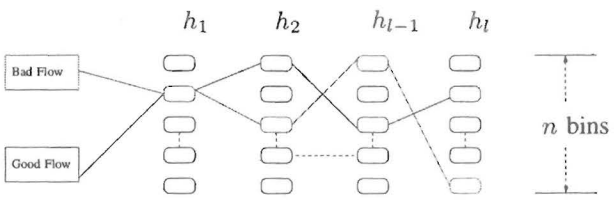



Fig. 8. Illustration of Bloom filter (adapted from [10]). There are $l$ stages with $n$ accounting bins in each stage. A hash function at each stage maps the flow id of the pack to one of the bins in that stage.

Bloom filters were introduced by Burton H. Bloom in 1970 as an efficient way of doing spell checking. We borrow some ideas of Bloom filters from Stochastic Fair BLUE [10] in reducing the complexity of SREQM based schemes. The basic idea behind a Bloom filter is given in Figure 8. There are $l$ hash stages that operate in parallel with $n$ accounting bins in each stage. Each bin in every stage also maintains a state-variable indicated by $H_j^i(\cdot)$ for the $j^{\text{th}}$ bin in the $i^{\text{th}}$ stage.

When a packet arrives, the flow id of the packet is hashed into one of the $n$ bins in each of the $l$ stages by a hashing function at each stage. The hashing function at stage $i$ is indicated by $h_i$. When a packet is mapped to one of the bins in each stage $i$, the $H_j^i(\cdot)$ corresponding to the bin to which the packet is mapped, is modified as $H_j^i(t) = H_j^i(t-1)\left(1-\frac{1}{T}\right)+\frac{1}{T}$, while the $H_j^i(\cdot)$ of all the other bins in the $i^{\text{th}}$ stage is modified as $H_j^i(t) = H_j^i(t-1)\left(1-\frac{1}{T}\right)$. This procedure is carried out for all the stages. If the state-variable associated with all the bins to which the packet is mapped is greater than a threshold $K$, then the packet is dropped; otherwise, it is added to the queue. The value of $K$ is dynamically modified, based on the aggregate queue occupancy, as we do in previous algorithms.

Usually the number of bins that we can afford in each stage is much smaller than the number of flows, and therefore, many flows may map to the same bin and share the same state-variable associated with that bin. This can cause misclassification. That is, a responsive flow may be mapped to the same bin as a non-responsive flow and share the same state variable. Since the non-responsive flow is sending at a very high rate, the common state-variable for both the flows will exceed the threshold $K$, resulting in packet drops from both the flows. The purpose of multiple hash stages is to reduce misclassification. By using multiple hash stages and selecting appropriate hash functions for each stage, we can make sure that a flow is not misclassified at least in some stages. Multiple hash stages attenuate the probability of misclassification exponentially.

Let there be $l$ hash stages with $n$ accounting bins per stage. Suppose that there are $m$ "bad" flows in total. Using simple analysis, we can show that probability that a good flow shares a bin with a bad flow in all the $l$ levels is

$$\mathbf{P}\{\text{A good flow shares all the bins with the bad flow}\} \approx \left(1-e^{-\frac{m}{n}}\right)^l. \quad (20)$$

Given the number of bad flows and the maximum probability of misclassification that we can tolerate, we can easily compute the number of stages required to design the Bloom filter from the above equation to achieve the required accuracy. Figures 9 and 10 plot the probability of misclassification, as a function of the number of non-responsive flows for different number of hash stages with 64 bins per stage and 256 bins per stage, respectively. We can readily see that increasing the number of stages or the number of bins per stage can decrease the probability of misclassification considerably.

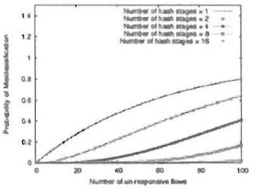


Fig. 9. Probability of misclassification with 64 bins per stage. As we increase the number of stages from 1 to 16, the probability of misclassification decreases.

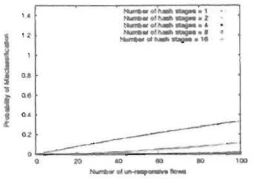


Fig. 10. Probability of misclassification with 256 bins per stage. As we increase the number of stages from 1 to 16, the probability of misclassification decreases.

## VI. Conclusion and Future Work

In this paper we proposed two efficient congestion control algorithms that can protect responsive traffic from non-responsive traffic and also guarantee QoS among flows. The performance of the algorithms were studies under different scenarios. We also showed that by using techiques such as sampling and Bloom filters, we can reduce the implementation complexity to a great extend. Future work includes improving the algorithm to guarantee QoS in very short time-scales, in the order of every packet cycle.

## References


[1] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless Fair Queueing: Achieving Approximately Fair Bandwidth Allocation in High Speed Networks," in *Proc. of ACM SIGCOMM'98*, Sep 1998, pp. 118–130.

[2] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker, "Approximate Fairness Through Differential Dropping," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 2, pp. 23–39, 2003.

[3] V. Ramaswamy, "Efficient Control of Non-Cooperative Traffic Using Sending Rate Estimate-Based Queue Management Schemes," Ph.D. dissertation, The University of Mississippi, 2006.

[4] V. Ramaswamy and S. Eidenbenz and L. Cuellar and N. Hengartner, "Preventing Bandwidth Abuse at the Router Through Sending Rate Estimate-based Active Queue Management," in *To appear in Proc. of IEEE ICC'07*, June 2007.

[5] A. Das, D. Dutta, A. Goel, A. Helmy, and J. Heidemann, "Low State Fairness: Lower Bounds and Practical Enforcement," in *Proc. of IEEE INFOCOM'05*, 2005, pp. 2436–2446.

[6] A. S. Sedra and K. C. Smith, *Microelectronic Circuits*. Oxford University Press, June 1997.

[7] V. Ramaswamy and S. Eidenbenz and L. Cuellar and N. Hengartner, "Light-Weight Control of Non-Responsive Traffic with Low Buffer Requirements," in *To appear in Proc. of IFIP Networking'07*, May 2007.

[8] J. F. Hayes and T. V. J. G. Babu, *Modeling and Analysis of Telecommunications Networks*. Wiley-Interscience, 2004.

[9] B. Bloom, "Space/time Trade-Offs in Hash Coding With Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, July 1970.

[10] W. Feng, D. Kandlur, D. Saha, and K. Shin, "Stochastic Fair BLUE: A Queue Management Algorithm for Enforcing Fairness," in *Proceedings of IEEE INFOCOM'01*, April 2001, pp. 1520–1529.