This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2007 proceedings.

Small active counters

Rade Stanojević
Hamilton Institute, NUIM, Ireland
rade.stanojevic@nuim.ie

***Abstract*— The need for efficient counter architecture has arisen for the following two reasons. Firstly, a number of data streaming algorithms and network management applications require a large number of counters in order to identify important traffic characteristics. And secondly, at high speeds, current memory devices have significant limitations in terms of speed (DRAM) and size (SRAM). For some applications no information on counters is needed on a per-packet basis and several methods have been proposed to handle this problem with low SRAM memory requirements. However, for a number of applications it is essential to have the counter information on every packet arrival. In this paper we propose two, computationally and memory efficient, randomized algorithms for approximating the counter values. We prove that proposed estimators are unbiased and give variance bounds. A case study on Multistage Filters (MSF) over the real Internet traces shows a significant improvement by using the active counters architecture.**

***Index Terms*— Counter architecture, high-speed measurements, router, data streaming algorithms.**

I. Introduction

Statistics counters are an essential element of most of data streaming or sampling algorithms. Maintaining them at high line speeds is a challenging research problem. Briefly, we need an architecture (1) to store a large number of counters (say millions) and (2) to update a large number (say tens of millions) of counters per second. While chip and large Dynamic RAM (DRAM) can easily satisfy condition (1), DRAM access times of $50 - 100ns$ cannot accommodate large number of counter-updates needed[1]. On the other hand, expensive Static RAM (SRAM) with access times of $2 - 6ns$ allow much more counter increments per second, but are not economical for large number of counters. Recently, several solutions [16], [22], [17], [18] are proposed that use hybrid SRAM/DRAM architecture: for each counter m-bit counter ($m \ll 64$) is stored in SRAM while full counter is stored in DRAM. SRAM counters are updated on per-packet basis and when some of them come close to overflow different Counter Management Algorithms (CMA) decide which counters should be flushed to DRAM. These approaches share a common feature: it is not possible to estimate counter value without accessing DRAM. We use the term *Passive Counters* to refer to such schemes. For a number of applications it is essential to have an estimate of the counter on every packet arrival. For example, in Multistage Filters (MSF) [6], the conservative update step requires knowledge of counters at each stage, and is essential for good performance of MSF: [6] reports up to 100 times higher false-positive ratio without conservative updates. Similarly, in Smart Sampling [3], the sampling probability depends on the current estimate of the flow size. The algorithm for building a Spectral Bloom Filter [2], the algorithm for online hierarchical-heavy-hitter identification [21] as well as efficient implementations of hash tables [19] also require the volume estimates on per-packet basis. We will use term *Active Counters* for schemes that allow estimate of full counters on per-packet basis without DRAM access. Thus active counters have to store enough information in fast memory. Storing the full-width counter in SRAM is an example of an active counter. At high-speeds these counters would require up to 64 bits to prevent overflow. The main objective of this paper is the design of small active counter architecture with small errors between the small-counter-based estimate and the real volume. Having efficient small active counters would reduce SRAM space needed for a number of applications; or equivalently, for given SRAM space one can fit more small active counters than regular 32(or 64)-bit counters, which in turn can significantly improve performance of data streaming algorithms with limited memory.

[1]Moreover one cannot expect in the near future that the speeds of DRAM (which increase approximately 7% per year) will reach the speeds of backbone links (which roughly double every 18 months)[6], [1].

A. Related work

The design of efficient statistics counter architecture has been identified as an important problem by Shah et al [18]. They propose hybrid SRAM/DRAM architecture in which DRAM is used to store the statistics counter while a small SRAM is used to enable counter updates at line rate. In other words, suppose that we need to track n counters of size M bits, then n counters of full size M bits are stored in DRAM and n counters of a smaller size $m < M$ bits are cached in SRAM. The counters in SRAM are updated on every packet arrival to keep track of the recent history of counter updates and are periodically flushed to corresponding DRAM. The decision as to which counter should be flushed to DRAM is made by a Counter Management Algorithm (CMA). By maintaining smaller SRAM counters required, SRAM space is reduced, while larger access times of DRAM are compromised by accessing DRAM not too often. The size of SRAM counters, as well as the frequency of DRAM access is determined by CMA. CMA decides which counters should be updated first. Different proposals have different CMAs. The following features are shared by previous proposals:

- *full counters are stored in slow DRAM.*
- *counters stored in DRAM are exact.*

In order to provide the exact values of DRAM counters

CMA must ensure that no SRAM counter overflows before updating to DRAM.

Shah et al [18] propose a CMA called Largest Counter First (LCF). In LCF, SRAM counter with the largest value is flushed to DRAM. LCF is difficult to implement at high speeds for the following two reasons. First, LCF requires a large amount of control memory: while LCF SRAM counters are 9 bits wide control memory can take up to 20 bits per counter. Second, LCF requires finding the largest of n counter values which is the main performance bottleneck of LCF.

In order to reduce control memory consumption of LCF and expensive search for the largest counter value, Ramabhadran and Varghese [16] proposed a new CMA called Largest Recent with threshold T, $LR(T)$. Briefly, $LR(T)$ tracks the list of counters with values greater than the threshold T and makes the decision as to which SRAM counter should be flushed to DRAM using only this list. By doing this, the computational complexity of LCF is significantly reduced, while the usage of the efficient data structure called aggregated bitmap requires only 2 bits per counter devoted for the control memory. Compared to LCF (that consumes 29 bits per counter) $LR(T)$ requires only 11 bits per counter.

Roeder and Lin [17] developed an extension to LCF and $LR(T)$ to reduce the memory requirements for nonuniform traffic patterns. Namely, if the frequency of updating fast memory counters is not uniform over counters, then allocating more bits to SRAM counters with more frequent updates would reduce the number of DRAM accesses; therefore the amount of fast memory space needed for storing recent history counters can be reduced. They propose multiple levels of fast memory instead of one: one level of SRAM and one or more levels of CAM (Content Addressable Memory) for storing the partial counter values. Their enhancement can reduce the amount of equivalent fast memory storage up to 28%.

A recent paper by Zhao et al [22] removed the basic design objective of previous approaches that does not allow SRAM-counter overflows. The key observation is, assuming that SRAM counters are large enough, counter overflows occur rarely enough so that a small buffer for storing this overflowed counters would be sufficient to allow the exact tracking of DRAM counters with extremely low probability of data loss[2]. The implementation of their proposal requires $l(\mu) = \lceil \log_2(\frac{1}{\mu}) \rceil$ bits for SRAM counters, and $\epsilon < 1$ (say 0.01) bits per counter for buffer that keep track of overflowed counters. Here μ represents the ratio between the access times of SRAM and DRAM. For currently available SRAM/DRAM devices, μ is in the range $[1/50, 1/10]$ which implies $l(\mu) \in \{4, 5, 6\}$. We believe that the multilevel SRAM/CAM/DRAM architecture, similar to those proposed in [17], can further reduce equivalent fast memory storage, but this is out of scope of the present paper.

[2]To quote[22] "However, in practice there is no need to worry about this probability, since it can be made so small that even if router operates continuously for billions of years (say from Big Bang to now)), the probability that a single loss of increment happens is less than 1 over a billion".

B. Our contributions

The current proposals for efficient counter statistics architecture belong to class that we call passive counters: full counter information is stored in DRAM, and it is not possible to estimate the full counter value without accessing DRAM. While for many applications passive counters are good enough [11], [12], for a number of applications need for full counter value, or its estimate, is required on per-packet basis [6], [2], [3], [21]. In these cases, at high speed links (say 10Gbps or more), it is necessary to have enough information for estimation of full counter based on information stored in the fast memory. The design of efficient active counter architecture is the main concern of this paper.

Briefly, the main contributions of this paper are the following:

- Two active counter architecture schemes, SAC and HAC, are proposed that have small per counter memory requirements, yet small expected error.

- The proof of unbiasedness of presented estimators and the analysis of the expected error.

- The framework for the resource control of SAC and HAC, and a demonstration of the potential benefit of proposed solutions is presented in a case study done with Multistage filters over real Internet traces.

Both of our schemes are randomized and therefore give the estimate of total traffic rather than the exact values. Our first scheme called SAC use only SRAM memory, with m bits per counter devoted for tracking values in range $[0, 2^M]$ where $M > m$. The standard error of this scheme is proportional to $\frac{1}{\sqrt{2^k+2^{k-r}}}$, for some $r < k < m$. In order to reduce the error we developed the second scheme called HAC which uses hybrid SRAM/DRAM architecture. In HAC the main counter information is stored in SRAM, while DRAM is used for storing augmenting counters which are used to reduce the standard error. Our analysis shows that if η is the relative frequency of accessing of DRAM, and $\hat{c}$ the estimate of the counter, then the standard error of HAC is proportional to $\frac{\eta}{\sqrt{\hat{c}}}$ (which can be substantially smaller for large counter values).

We argue that in applications of statistics counters for firewall support, intrusion detection, performance monitoring, flow control, packet schedulers, load balancing or traffic engineering, no need for exact counters exist. We believe that small errors (of say 1%) are acceptable, so that all reasons for design of existing passive counters apply to active counter architecture as well. We also believe that storing counter information in on-chip SRAM can be beneficial for remote monitoring/control as it would reduce latency of SMNP queries at heavily utilized network processors that usually have low priority (and therefore high latency).

Throughout this paper we use the following notation. For a real number x: by $\lfloor x \rfloor$ we denote the largest integer not greater than x by $\lceil x \rceil$ we denote the smallest integer not less than x and by $\{x\} = x - \lfloor x \rfloor$ we denote the residuum of x modulo

```
1   UpdateCounter(i, inc)
2     A[i] = A[i] + ⌊inc / 2^{r·mode[i]}⌋
3     With probability {inc / 2^{r·mode[i]}}: A[i] = A[i] + 1
4     if A[i] > 2^k
5       mode[i] = mode[i] + 1;
6       p = {A[i] / 2^r};
7       A[i] = ⌊A[i] / 2^r⌋;
8       With probability p: A[i] = A[i] + 1;
9     endif
10    if mode[i] == 2^l
11      RenormalizeCounters(r)
12      r = r + 1;
13    endif

14  Initialize()
15    r = 1;
16    mode[i] = 0;
17    A[i] = 0;

18  RenormalizeCounters(r)
19    for s = 1 : n
20      old_mode = mode[i];
21      mode[i] = ⌈mode[i] · r/(r+1)⌉;
22      r_1 = mode[i] · (r + 1) − old_mode · r;
23      p = {A[i] / 2^{r_1}};
24      A[i] = ⌊A[i] / 2^{r_1}⌋;
25      With probability p: A[i] = A[i] + 1;
26    endfor
```

Fig. 1. The pseudocode of SAC.

1.

II. Simple Active Counters (SAC) scheme

Suppose that we have $q \cdot n$ bits of SRAM allocated for n counters. Let V_i be real value of counter i. Our goal is to develop scheme that estimates V_i using only q bits of SRAM memory. Our scheme works as follows. We divide the available q bits of the i-th SRAM counter in two parts: l-bit exponent part that we call $mode[i]$, and the k-bit estimation part $A[i]$. We use the estimator

$$\hat{V}_i = A[i] \cdot 2^{r \cdot mode[i]} \tag{1}$$

Here r is the global parameter, and is determined by scale over which this set of counters run. To update $A[i]$ we use a randomized scheme. Suppose that we need to increment the counter i by value inc. If $inc \geq 2^{r \cdot mode[i]}$ then we first increment $A[i]$ by $\lfloor \frac{inc}{2^{r \cdot mode[i]}} \rfloor$ and then increment $A[i]$ by 1 with probability given by normalized value of the residue: $\frac{1}{2^{r \cdot mode[i]}}(inc - \lfloor \frac{inc}{2^{r \cdot mode[i]}} \rfloor \cdot 2^{r \cdot mode[i]})$. Similarly, for $inc < 2^{r \cdot mode[i]}$ we increment $A[i]$ by 1 with probability $\frac{inc}{2^{r \cdot mode[i]}}$.

The pseudocode of the scheme is given in Figure 1. Renormalization step moves counters to a higher scale: $r = r + 1$.

Initially, we statically divide the available memory of q bits in two parts of size l and k such that $l + k = q$. How to chose "right" allocation pair (k, l) depends on various factors, and we will discuss this in detail in Section V.

The analysis from Section IV shows that the standard error of the scheme is approximately proportional to $\frac{1}{\sqrt{2^k + 2^{k-r}}}$. This error can be substantial for very small k. Intuitively, for large counters, the frequency of updates is very small, which implies the large variance of the background estimator. In the next section we present the architecture that will use augmenting DRAM counters to reduce the variance of estimators.

```
1   UpdateCounter(i, inc)
2     if inc / 2^{r·mode[i]} > 1/η
3       A[i] = A[i] + ⌊inc / 2^{r·mode[i]}⌋
4       With probability {inc / 2^{r·mode[i]}}: A[i] = A[i] + 1
5     else
6       With probability 1/η do
7         B[i] = B[i] + inc;
8         if B[i] ≥ 2^{r·mode[i]} / η;
9           A[i] = A[i] + 1;
10          B[i] = B[i] − 2^{r·mode[i]} / η;
11        endif
12      enddo
13    endelse
14    if A[i] > 2^k
15      mode[i] = mode[i] + 1;
16      B[i] = RandomUniform[0, 2^{r·mode[i]} / η]
17      p = {A[i] / 2^r};
18      A[i] = ⌊A[i] / 2^r⌋;
19      With probability p: A[i] = A[i] + 1;
20    endif
21    if mode[i] == 2^l
22      RenormalizeCounters(r)
23      r = r + 1;
24    endif

25  Initialize()
26    r = 1;
27    mode[i] = 0;
28    A[i] = 0;
29    B[i] = 0;

30  RenormalizeCounters(r)
31    for s = 1 : n
32      old_mode = mode[i];
33      mode[i] = ⌈mode[i] · r/(r+1)⌉;
34      r_1 = mode[i] · (r + 1) − old_mode · r;
35      p = {A[i] / 2^{r_1}};
36      A[i] = ⌊A[i] / 2^{r_1}⌋;
37      With probability p: A[i] = A[i] + 1;
38      B[i] = RandomUniform[0, 2^{(r+1)·mode[i]} / η];
39    endfor
```

Fig. 2. The pseudocode of HAC.

III. Hybrid Active Counters (HAC) scheme

Suppose again that we have q bits per counter in SRAM, divided in two parts with sizes of k ($A[i]$) and l ($mode[i]$) bits. Assuming knowledge of the full counter value V_i, setting $A[i]$ to $round(\frac{V_i}{2^{r \cdot mode[i]}})$ would imply the lowest errors under this circumstances. However, knowledge of the exact value V_i does not exist, so we have to find other ways to capture the top k bits of V_i. As we noticed, purely SRAM solution exhibit $O(\frac{1}{\sqrt{2^k}})$ errors. This is due to high variance of the background estimators. To illustrate this, consider the following example: scale $r = 2$, $mode[1] = 5$, with all increments equal to 1. In this setting, $A[i]$ is incremented by 1 with probability 2^{-10}. Suppose that the counter V_i updates come with a rate of 1 per unit of time, this means that the time between the two consecutive $A[i]$ increments is given by the geometric random variable $G(p_0)$, with $p_0 = 2^{-10}$. The expected value of $G(p_0)$ is $1/p_0 = 1024$, while the variance of $G(p_0)$ is $\frac{1-p_0}{p_0^2} \approx 2^{20}$. Suppose now that we have available additional memory which can be accessed only occasionally, say once in 16 time units on average. Let $B[i]$ be the additional memory space and in the every time unit increment $B[i]$ by 1 with probability $p_1 = 1/16$. By incrementing $A[i]$ when $B[i]$ reaches 64, the time between two consecutive $A[i]$ increments is given by the random variable Z_1 which is sum of 64 i.i.d. geometric

random variables $G(p_1)$. The expected value of Z_1 is $64 \cdot \frac{1}{p_1} = 1024$, while the variance of Z_1 is $64 \cdot \frac{1-p_1}{p_1^2} \approx 2^{14}$, which is approximately 2^6 times smaller than in the case without additional memory. We can think of $B[i]$ as of additional counters stored in the slower DRAM with rare accesses.

Our second scheme HAC formalizes the reasoning from the simple example presented above. The basic SRAM architecture is the same as in SAC: a memory of $n \cdot q$ bits; each q bits counter is divided in k bits estimate space $A[i]$ and l bits $mode[i]$. The difference between SAC and HAC is in the way they increment $A[i]$. In HAC, each of n counters have a shadow counter in DRAM, $B[i]$. Let η be the allowable relative DRAM frequency: every η SRAM accesses we can have one DRAM access. We update $B[i]$ with probability $\frac{1}{\eta}$, and increment $A[i]$ when $B[i]$ "overflows".

Figure 2 contains the pseudocode of HAC. For low r and $mode[i]$ (if statement in line 2 is positive) there is no need for DRAM, which become beneficial only when $r \cdot mode[i]$ is larger.

At low speeds, η can be equal to 1, which means that we can store full counters in DRAM. However, in our case of interest, $\eta > 1$, for the moment we will assume that is manually configurable constant. In Section V we will present a possible control strategy for η, that will take into consideration the available system bus and DRAM bandwidth. In the case of constant η no need for the renormalization of DRAM counters $B[i]$ exists. However, in the case of dynamic η, the renormalization of $B[i]$ is necessary.

Compared to other existing counter architectures HAC does not have exact counter value, but stores only its estimate. Note a paradigm shift: *the main counter information is stored in SRAM while DRAM is used to store information only from the recent history.*

IV. Analysis

In this section we provide the analysis of the proposed schemes: SAC and HAC. Throughout this section we concentrate on a single counter, say $(A[1], mode[1])$, and denote it with $(A, mode)$. Let $u_1, u_2, \ldots$ be the sequence of increments of counter $(A, mode)$. Denote $V_j = \sum_{s=1}^{j} u_s$. For each $j = 1, 2, \ldots$, denote by A_j the k-bit random variable A after j updates, by $mode_j$ we denote the random variable $mode$ after j updates, and by r_j the random variable that determines scale r after j updates. In the next theorem we show that the following estimator of V_j is unbiased:

$$\hat{V}_j = A_j \cdot 2^{mode_j \cdot r_j}.$$

Theorem 1: The SAC estimator $\hat{V}_j$ is unbiased:

$$E[\hat{V}_j] = V_j.$$

Proof: Denote by $(A'_j, mode'_j, r'_j)$,$(A''_j, mode''_j, r''_j)$ and $(A'''_j, mode'''_j, r'''_j)$ the values of $(A, mode, r)$ after execution of the lines 3, 9 and 13 of the SAC algorithm given in Figure 1 (Recall that index i is omitted, and that $inc = u_{j+1}$). We have that

$$E[A'_j \cdot 2^{r'_j \cdot mode'_j} | (A_j, mode_j, r_j)] = \left(A_j + \lfloor \frac{u_{j+1}}{2^{r_j \cdot mode_j}} \rfloor\right) \cdot 2^{r_j \cdot mode_j} + \{\frac{u_{j+1}}{2^{r_j \cdot mode_j}}\} \cdot 2^{r_j \cdot mode_j} = A_j \cdot 2^{r_j \cdot mode_j} + u_{j+1}.$$

Thus

$$E[A'_j \cdot 2^{r'_j \cdot mode'_j}] = E[A_j \cdot 2^{r_j \cdot mode_j}] + u_{j+1}. \quad (2)$$

After the line 9 is executed, we have that either $(A''_j, mode''_j, r''_j)$ is equal to $(A'_j, mode'_j, r'_j)$ or $mode''_j = mode'_j + 1$ and

$$E[A''_j \cdot 2^{r''_j \cdot mode''_j} | (A'_j, mode'_j, r'_j)] = \left\lfloor \frac{A'_j}{2^{r'_j}} \right\rfloor \cdot 2^{r'_j \cdot (mode'_j + 1)} + \left\{ \frac{A'_j}{2^{r'_j}} \right\} \cdot 2^{r'_j \cdot (mode'_j + 1)} = A'_j \cdot 2^{r'_j \cdot mode'_j}.$$

Thus, in both subcases

$$E[A''_j \cdot 2^{r''_j \cdot mode''_j}] = E[A'_j \cdot 2^{r'_j \cdot mode'_j}]. \quad (3)$$

Similarly

$$E[A'''_j \cdot 2^{r'''_j \cdot mode'''_j}] = E[A''_j \cdot 2^{r''_j \cdot mode''_j}]. \quad (4)$$

Since $(A'''_j, mode'''_j, r'''_j)$ is equal to $(A_{j+1}, mode_{j+1}, r_{j+1})$, from (2), (3) and (4) we conclude that

$$E[\hat{V}_{j+1}] = E[A_{j+1} \cdot 2^{r_{j+1} \cdot mode_{j+1}}] = E[A_j \cdot 2^{r_j \cdot mode_j}] + u_{j+1} = E[\hat{V}_j] + u_{j+1}.$$

Now by a straightforward mathematical induction argument the assertion of the theorem follows. ■

The following theorem establishes the same result for the HAC.

Theorem 2: The HAC estimator $\hat{V}_j$ is unbiased:

$$E[\hat{V}_j] = V_j.$$

Proof: Similarly, as in proof of Theorem 1 we denote by $(A'_j, mode'_j, r'_j)$,$(A''_j, mode''_j, r''_j)$ and $(A'''_j, mode'''_j, r'''_j)$ values of $(A, mode, r)$ after the execution of the lines 13, 20 and 24 of the HAC algorithm (given in Figure 2) respectively; $inc = u_{j+1}$.

From the proof of the Theorem 1 we have that the relations (3) and (4) are satisfied. Now we prove that relation (2) is satisfied as well. We distinguish two cases:

1st Case: $\frac{u_{j+1}}{2^{r \cdot mode[i]}} > \frac{1}{\eta}$. Then HAC is identical to SAC and (2) follows.

2nd Case: $\frac{u_{j+1}}{2^{r \cdot mode[i]}} \leq \frac{1}{\eta}$. First, note that if B is a random variable uniformly distributed in the segment $[0, a]$ then for any real number x, the random variable $mod(B + x, a) = B + x - a \cdot \lfloor \frac{B+x}{a} \rfloor \in [0, a]$ is also uniformly distributed in $[0, a]$. This means that if we initialize the DRAM counter B with the uniform distribution, it will remain uniform (in the appropriate segment $[0, \frac{2^{r_j \cdot mode_j}}{\eta}]$) after the updates (line 10). This implies that the probability of $B + u_{j+1} > \frac{2^{r_j \cdot mode_j}}{\eta}$ is given by $\frac{u_{j+1}}{\frac{2^{r_j \cdot mode_j}}{\eta}}$. Thus counter A_j is incremented by one (line 9) with probability

$$q_0 = \frac{1}{\eta} \cdot \frac{u_{j+1}}{\frac{2^{r_j \cdot mode_j}}{\eta}} = \frac{u_{j+1}}{2^{r_j \cdot mode_j}}.$$

Formally

$$E[A'_j \cdot 2^{r'_j \cdot mode'_j} | (A_j, mode_j, r_j)] = (A_j + q_0) \cdot 2^{r_j \cdot mode_j} = \\ = A_j \cdot 2^{r_j \cdot mode_j} + u_{j+1}.$$

Which implies

$$E[A'_j \cdot 2^{r'_j \cdot mode'_j}] = E[A_j \cdot 2^{r_j \cdot mode_j}] + u_{j+1}.$$

■

A. Variance estimation

Suppose that a counter runs in scale r and mode $m > 0$. Then by definition the value A of k-bit counter is in the range $[2^{k-r}, 2^k - 1]$. Denote by $T(A, m)$ the random variable that represents the amount of total traffic needed from the start of counting until the SAC (HAC) counter reaches the state (A, m). In this section we evaluate the variance of $T(A, m)$, assuming an uniform increment size θ.

Theorem 3: Suppose that increments to the SAC counter are uniform and given by $\theta > 0$. Then, for the random variable $T(A, m)$ defined above we have:

$$E[T(A, m)] = A \cdot 2^{rm} \tag{5}$$

and

$$Var[T(A, m)] = 2^{k-r} \left(1 + \frac{2^{2rm} - 1}{2^r + 1} - \theta(2^{rm} - 1))\right) + \\ + (A - 2^{k-r}) \left(2^{2rm} - \theta 2^{rm}\right). \tag{6}$$

Proof: Recall that for a geometric random variable[3] $G(p)$: $E[G(p)] = \frac{1}{p}$ and $Var[G(p)] = \frac{1-p}{p^2}$. For each s denote by $W(r, s)$ the random variable that represents the amount of traffic needed to make one increment of the counter $(A, mode)$ when $mode = s$. The probability of incrementing the counter $(A, mode)$ in a single trial is $p_s = \frac{\theta}{2^{rs}}$. Therefore, the number of trials before the increment of $(A, mode)$ is $G(p_s)$ and since each trial corresponds to θ of the total traffic we have

$$W(s) = \theta G(p_s).$$

Time from the beginning of counting can be divided in $m + 1$ intervals, corresponding to each mode $s = 0, 1 \ldots, m$. In mode $s = 0$ A is incremented $q_0 = 2^k$ times, for modes $s = 1, 2 \ldots, m - 1$, we have $q_s = 2^k - 2^{k-r}$ increments, while in mode $s = m$ until the k-bit counter is A the number of its increments is $q_m = A - 2^{k-r}$. Denote by $Q(s)$ the total arrived traffic in each of these $m + 1$ intervals. Now for any $s = 0, 1, \ldots, m$

$$Q(s) = \sum_{j=1}^{q_s} W_j(s),$$

where $W_i(s)$ is the set of independent identically distributed (i.i.d.) random variables, with distribution given by $W(s)$. Therefore:

$$E[Q(s)] = \sum_{j=1}^{q_s} E[W_j(s)] = q_s \cdot \theta \frac{1}{p_s} = q_s \cdot 2^{rs}.$$

[3]Geometric random variable represents the number of Bernoulli trials needed for the first success, when the success of each trial has probability p.

and

$$Var[Q(s)] = \sum_{j=1}^{q_s} Var[W_j(s)] = q_s \cdot \theta^2 \frac{1 - p_s}{p_s^2} = q_s \cdot 2^{2rs}(1 - p_s).$$

Now $T(A, m)$ is given by:

$$T(A, m) = \sum_{s=0}^{m} Q(s).$$

The expected value of $T(A, m)$ is now:

$$E[T(A, m)] = \sum_{s=0}^{m} E[Q(s)] = q_0 + \sum_{s=1}^{m-1} q_s \cdot 2^{rs} + q_m \cdot 2^{rm} = \\ = 2^k + (2^k - 2^{k-r}) \sum_{s=1}^{m-1} \cdot 2^{rs} + (A - 2^{k-r}) \cdot 2^{rm} = \\ = 2^{k-r} + (2^k - 2^{k-r}) \sum_{s=0}^{m-1} \cdot 2^{rs} + (A - 2^{k-r}) \cdot 2^{rm} = . \\ = 2^{k-r} + (2^k - 2^{k-r}) \frac{2^{mr} - 1}{2^r - 1} + (A - 2^{k-r}) \cdot 2^{rm} = A \cdot 2^{rm}.$$

And the variance

$$Var[T(A, m)] = \sum_{s=0}^{m} Var[Q(s)] = q_0 + \sum_{s=1}^{m-1} q_s \cdot 2^{2rs}(1 - p_s) + \\ + q_m \cdot 2^{2rm}(1 - p_m) = 2^k + (2^k - 2^{k-r}) \sum_{s=1}^{m-1} \cdot 2^{2rs}(1 - \frac{\theta}{2^{rs}}) + \\ + (A - 2^{k-r}) \cdot 2^{2rm}(1 - \frac{\theta}{2^{rm}}) = 2^{k-r} + \\ + (2^k - 2^{k-r}) \sum_{s=0}^{m-1} \cdot (2^{2rs} - \theta 2^{rs}) + (A - 2^{k-r}) \cdot (2^{2rm} - \theta 2^{rm}) = \\ = 2^{k-r} + 2^{k-r}(2^r - 1)(\frac{2^{2mr} - 1}{2^{2r} - 1} - \theta \frac{2^{mr} - 1}{2^r - 1}) + \\ + (A - 2^{k-r}) \cdot (2^{2rm} - \theta 2^{rm}),$$

and (6) follows. ■

With $\delta(T(A, m))$ we denote the *Coefficient of variation*:

$$\delta(T(A, m)) = \sqrt{E\left[\left(\frac{T(A, m) - A2^{rm}}{A2^{rm}}\right)^2\right]} = \sqrt{\frac{Var[T(A, m)]}{(E[T(A, m)])^2}}$$

The following corollary characterizes the asymptotic behaviour of $\delta_{SAC}(T(A, m))$ at the beginning of mode m: $A = 2^{k-r}$.

Corollary 1: For SAC counters at the beginning of mode m ($A = 2^{k-r}$):

$$\delta^2_{SAC}(T(A, m)) = \frac{1}{2^{k-r+2rm}} \left(1 + \frac{2^{2rm} - 1}{2^r + 1} - \theta(2^{rm} - 1)\right). \tag{7}$$

When $r \cdot m$ is large:

$$\delta_{SAC}(T(A, m)) \approx \sqrt{\frac{1}{2^k + 2^{k-r}}}.$$

Proof: The equality (7) follows directly from (5) and (6). When $r \cdot m$ is large

$$\delta_{SAC}(T(A,m)) \approx \sqrt{\frac{1}{2^{k-r+2rm}}\frac{2^{2rm}}{2^r+1}} = \sqrt{\frac{1}{2^k+2^{k-r}}}.$$

■

In HAC case we have the following

Theorem 4: Suppose that the increments to the HAC counter are uniform and given by $\theta > 0$ with relative DRAM access frequency η. Then, for the random variable $T(A,m)$ defined above it holds:

$$E[T(A,m)] = A \cdot 2^{rm} \tag{8}$$

and

$$Var[T(A,m)] = A \cdot 2^{rm} \cdot \theta(\eta - 1). \tag{9}$$

Proof: As in the proof of Theorem 3, we divide the time into $m+1$ intervals, each corresponding to one mode $s = 0, 1, \ldots, m$ and denote by $W(s)$ the amount of traffic needed to make one increment of the HAC counter at mode s. Thus, $W(s)$ is given by the amount of traffic needed for the DRAM counter B to overflow (ie. become larger than $\frac{2^{rs}}{\eta}$) which is equal to the sum of $b = \frac{2^{rs}}{\eta\theta}$) i.i.d. random variables $Z_i \equiv \theta G(\frac{1}{\eta})$. Therefore

$$W(s) = \sum_{i=1}^{b} Z_i.$$

For $W(s)$ we have

$$E[W(s)] = \sum_{i=1}^{b} E[Z_i] = b \cdot \theta \cdot \frac{1}{\frac{1}{\eta}} = 2^{rs}.$$

and

$$Var[W(s)] = \sum_{i=1}^{b} Var[Z_i] = b \cdot \theta^2 \cdot \frac{1-\frac{1}{\eta}}{\frac{1}{\eta^2}} = 2^{rs} \cdot \theta(\eta - 1).$$

Now, the amount of traffic $Q(s)$ that corresponds to mode s is given by the sum of q_s i.i.d. random variables $W_j(s)$ identically distributed as $W(s)$: $Q(s) = \sum_{j=1}^{q_s} W_j(s)$. Note that $Var[W(s)] = \theta(\eta-1) \cdot E[W(s)]$ which implies that

$$E[Q(s)] = q_s \cdot 2^{rs}$$

and

$$Var[Q(s)] = \theta(\eta-1) \cdot q_s \cdot 2^{rs} = \theta(\eta-1) \cdot E[Q(s)]. \tag{10}$$

Thus, for $T(A,m) = \sum_{s=0}^{m} Q(s)$, from the proof of Theorem 3 we conclude that (8) is satisfied. Now (9) follows from (8) and (10). ■

From the previous theorem we get:

$$\delta_{HAC}(T(A,m)) = \sqrt{\frac{A \cdot 2^{rm} \cdot \theta(\eta-1)}{(A \cdot 2^{rm})^2}} = \sqrt{\frac{\theta(\eta-1)}{A \cdot 2^{rm}}}.$$

Remark 1. The variance calculated above neglected the initial period when $\frac{inc}{2^{r \cdot mode[i]}} > \frac{1}{\eta}$, during which the HAC is identical to the SAC. Since the SAC error is smaller than the HAC error in this regime, we do not use the hybrid scheme until the end of this period. Thus, the real HAC error is strictly less than the HAC error presented above. However, the difference is small since the initial period takes a relatively short period of time.

Remark 2. Note that from the Cauchy-Schwartz inequality, for any real valued random variable X: $\sqrt{E(X^2)} \geq E(|X|)$. Taking X to be the relative error of $T(A,m)$, $X = \frac{T(A,m)-A2^{rm}}{A2^{rm}}$, the previous inequality implies that the expected relative error is not greater than $\delta(T(A,m))$:

$$E\left[\left|\frac{T(A,m)-A2^{rm}}{A2^{rm}}\right|\right] \leq \delta(T(A,m))$$

V. Resource control

In this section we discuss possible approaches for controlling two variables that determine accuracy of our schemes: k and η.

A. Allocating q bits in two parts.

The basic question is the following. Given q bits of available memory for a counter $(A, mode)$, how many bits should we allocate to A and $mode$? Allocating more bits to A gives less space for $mode$ and therefore implies a potentially larger scale r. Larger r implies larger errors for fixed k. Figure 3 depicts the graphs of $\delta_{SAC}(T(A,m))$ for $k = 8$ allocating bits for A, in three modes $r = 1, 2, 3$, in the SAC case. On the other hand, allocating too much space for storing $mode$ might waste precious space for A. Thus, for a single counter (of size q) one possible way to choose the values k and l is to pick k and $l = q-k$ for which $\delta(T(A(k), mode(k)))$ is minimized. Thus, assuming that the performance objective is minimizing the value of $\delta(T(A(k), mode(k)))$ then k should be dynamically updated to a value that minimizes $\delta(T(A(k), mode(k)))$.

In the case of multiple counters we should pick a common k for all counters. The performance objective can depend on various factors. For example some counters might require higher precision then others; for various applications counters with very low or very high values might be more important than others. In general, for n counters: $(A_i, mode_i)$, picking the optimal k is formally equivalent to the optimization problem:

$$\min \sum_{i=1}^{n} f_i(A_i(k), mode(k), \delta(T(A_i(k), mode(k)))). \tag{11}$$

Here f_i are cost functions. If none of the counters is prioritized by default, f_i is independent of i. One should be careful since changing k might cause change of scale r and all counters should be re-normalized to take into account the new conditions. This re-normalization can be done in a straightforward manner and will not be discussed here.

B. Controlling the η

In the HAC case an important parameter that determines accuracy of active counters is relative DRAM access frequency η. Lower η corresponds to better accuracy of the HAC counters. However, there exist a tradeoff between the usage of DRAM and the accuracy of the HAC scheme. The performance bottleneck in many network processors is

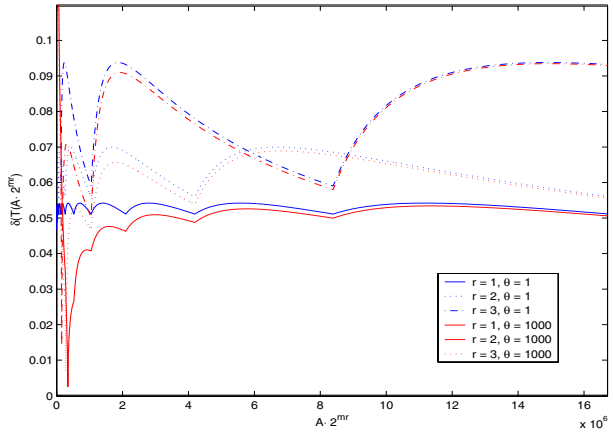

Fig. 3. Coefficient of variation $\delta_{SAC}(T(A,m))$ for $r = 1, 2, 3$ and $\theta = 1, 1000$; $k = 8$.

the system bus and DRAM bandwidth. From the point of exploitation of this resource we can distinguish two cases:

1. The devoted number of DRAM accesses used by HAC in a unit of time Δ (say 1 sec) is constant and is given by α.

2. DRAM bandwidth is shared between HAC and other applications.

In both cases we can isolate the buffer for storing the DRAM access queries required by HAC. The rate by which this buffer is serviced depends on the level of utilization and the DRAM bandwidth resource allocation algorithm. Bearing in mind that η is the parameter that controls the arrival rate to this buffer, we can exploit ideas from the large spectrum of AQM algorithms. For example we can control η based on queue length; by tracking the low pass filter of the queue length as in RED like schemes [9], [7], [8] we can dynamically update η to keep the queueing delay low. Virtual queue techniques can be used as well [10], [13], as well as other control strategies. For the stability of such control schemes, it is beneficial that there does not exist feedback delay unlike the case of TCP/AQM where feedback delays can be significant and dangerous from the stability point of view.

The HAC algorithm defined in Section III assumes invariant sampling rate η. In the context of variable η the augmenting DRAM counters $B[i]$ should take into account the sampling rate. To do this, we can track $\eta[i]$ - the sampling rate at the last update of i-th DRAM counter $B[i]$. At every update of the counter $B[i]$ we first re-normalize $B[i]$ taking into account the current η:

$$B[i] := B[i] \cdot \frac{\eta[i]}{\eta},$$

then reset $\eta[i]$ to the current value of η

$$\eta[i] := \eta.$$

After this, continue with updating the counter $B[i]$. By doing this unbiasedness of the appropriate counters is preserved. In normal conditions, measurements [14] show that, on typical 150Mbps+ links, basic IP parameters (such as the number of active connections, proportion of TCP traffic, aggregate IP traffic, etc.) do not change dramatically in short time periods which protects η from large oscillations.

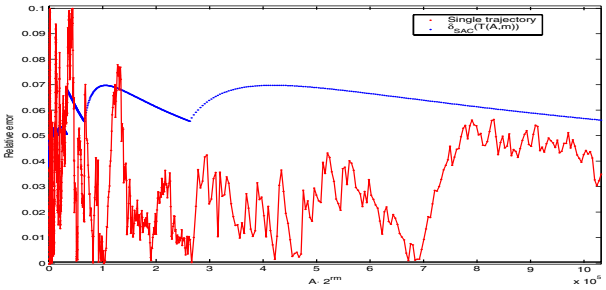

Fig. 4. Relative error for a single SAC with unit increments; $q = 12, k = 8$.

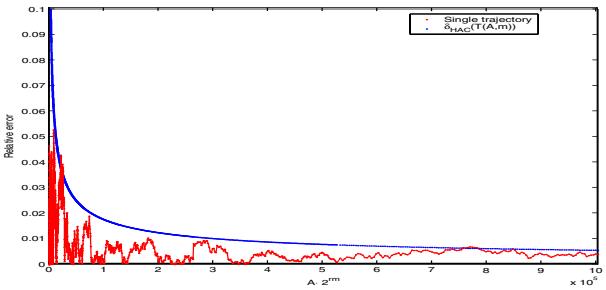

Fig. 5. Relative error for a single HAC with unit increments; $q = 12, k = 8$.

Remark 3. At highly loaded links, DRAM bandwidth and network processors are also highly utilized and therefore necessitate higher η. While the accuracy of HAC scheme decreases as η increases, at highly loaded links counter values grow as well which is beneficial for HAC accuracy. However, we are unable to quantify this tradeoff between link (DRAM bandwidth) utilization and HAC accuracy since it is a function of other network processor parameters which are hard to characterize.

Remark 4. A relatively small queue for storing DRAM updates can (in conjunction with appropriate queue management algorithm) ensure virtually zero loss of data (see [22]).

VI. Evaluation

In the first experiment[4] we evaluate the behavior of a single SAC (and HAC) counter over a stream of unit increments. The stream length is $L = 1000000$. Figures 4 and 5 depict observed relative errors of SAC and HAC respectively, together with the corresponding coefficient of variation. The counter size is $q = 12$ and $k = 8$.

Both SAC and HAC are designed to accept nonuniform increments. The technical report [20] contains experiments that show the behavior of both schemes under nonuniform increments given by a NLANR trace [15].

A. Errors versus counter size q

In this experiment we evaluate the average errors for SAC and HAC as a function of counter size q. We use

[4]All MATLAB simulations used in this section can be downloaded from http://www.hamilton.ie/person/rade/AC/

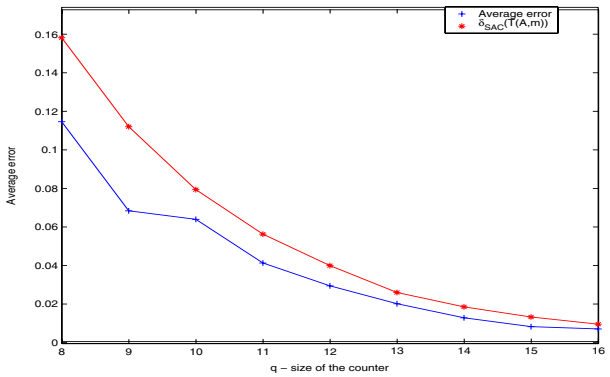

Fig. 6. SAC. Average errors for different values of q.

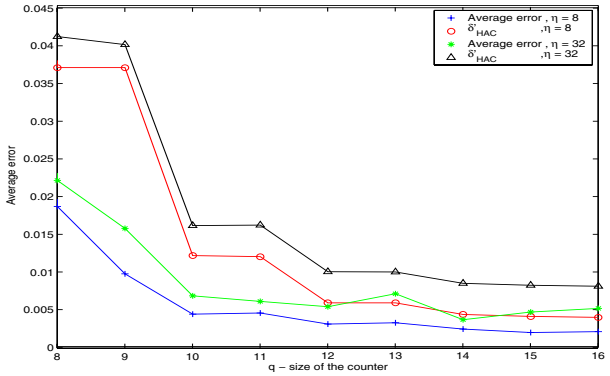

Fig. 7. HAC. Average errors for different values of q. Stream length 500000 unit increments

the cost function $f(A(k), mode(k), \delta(T(A(k), mode(k)))) = \delta(A(k), mode(k))$ for dividing q bits in two parts.

Figure 6 depicts the results based on a stream with unit increments in the SAC case. Averages are based on 10 independent runs. Figure 6 also contains the $\delta_{SAC}(T(A(k), mode(k)))$ for k that minimizes the cost function. As we already noticed (Remark 2, Section IV-A) the coefficient of variation δ is greater than the expected relative error.

In the HAC case the comparison between errors and counter size is not that straightforward as it depend on several other factors: η and the value of the counter $A \cdot 2^{r \cdot mode}$. In the SAC case the rounding error which is of order[5] $O(\frac{1}{2^k})$ is negligible compared to expected δ_{SAC} and we neglected it in computing the expected error δ_{SAC}. However, in HAC case δ_{HAC} can be much smaller and for estimating the error we use $\delta'_{HAC}(T(A, mode)) = \delta_{HAC}(T(A, mode)) + \frac{1}{A}$. Figure 7 depicts the errors of the HAC estimates after 500000 unit increments averaged over 10 independent runs, and two different η: 8 and 32.

B. Case study: MSF

In this section we investigate the effects that active counters could have on one of the state of the art algorithms for identification of heavy hitters: Multistage Filters (MSF)[6]. As we said, previously proposed passive counters [18], [16], [17], [22] cannot be used since they store full counter information in DRAM, which is too slow to track the conservative-update step of MSF. MSF uses memory divided in two parts. The first part contains d levels with b counters (of size of q bits) on each level, while the second part contains the hash table that keeps information on heavy hitters; we denote by FC the number of available flow entries and by S the size of a flow entry, in bits. We use S bits of a flow entry to track information on the flow size (S_1 bits) and the flow identifier + hash overhead ($S_2 = S - S_1$ bits). Table I determines the parameters for two cases: one without active counter enhancements (standard), and another with active counters (with AC).

MSF	q	S_1	S_2
Standard	24	40	24
With AC	9	16	16

TABLE I

PARAMETERS OF MSF COUNTER SIZE AND FLOW ENTRY SIZE.

Smaller sizes of counters allow more counters per stage and more flow entries. The price that must be paid is the inaccuracy caused by smaller counters. Here we show that the gain obtained by more counters outweight the inaccuracy of small counters. We run MSF over two unidirectional NLANR traces: MRA and FRG [15]. The average number of packets per second is 60K (MRA) and 69K (FRG). The average throughput is 285Mbps (MRA) and 428Mbps (FRG). The available SRAM memory is 40Kbit. We use the recommendation from [6] to determine other parameters of MSF (b,d and FC) in both cases. Since in both cases counters are not large enough (the largest is of order of magnitude of 10^7) to exploit the benefit of HAC for simplicity we use only SAC[6]. By using the cost function $f_i(A_i(k), mode_i(k), \delta_{SAC}(T(A_i(k), mode_i(k)))) = \delta_{SAC}(T(A_i(k), mode_i(k)))$ the optimization criterion stabilizes k at value $k_1 = 6$ for the basic 9-bit counters. For 16-bit (flow container) counters that track flow sizes the value of k that solves the same optimization problem is $k_2 = 12$.

The metric of interest is the average error for the set of flows indexed by the set I defined as in [6] by:

$$Err(I) = \frac{\sum_{i \in I} |est(f_i) - r(f_i)|}{\sum_{i \in I} r(f_i)} \qquad (12)$$

Here $r(f_i)$ is the exact size of flow f_i and $est(f_i)$ is the MSF estimate. By L_K we denote the set of top-K flows. We evaluate the average errors for the top-100, top-250, top-500 and top-1000 flows. The results are presented in Figures 8 and 9. Observe that the errors of top-100 flows are approximately the same both with and without SAC, while the ratio between the errors $Err(L_K)$ for MSF with standard counters and MSF with SAC grows as K grows: MSF with SAC gives up to 6 times smaller errors in measuring the top-1000 flows. In parallel we show the errors of MSF with SAC using half as much memory (20Kbit).

[5]We track the top k bits($A \sim 2^k$).

[6]Using HAC would give errors that are strictly less then using only SAC, but the difference is not significant in the present experiment, so we concentrate on SAC only.

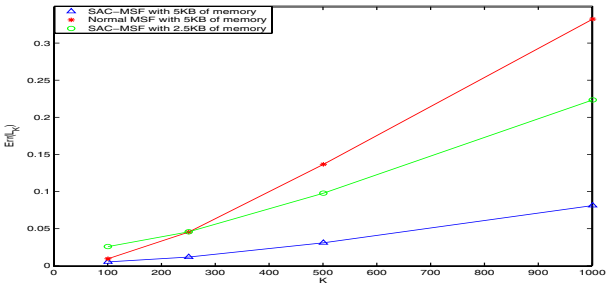

Fig. 8. MRA trace. The errors in estimation of the top K flows. MSF without any AC (with 40Kbit of memory), MSF + SAC (with 40Kbit of memory) and MSF + SAC (with 20Kbit of memory).

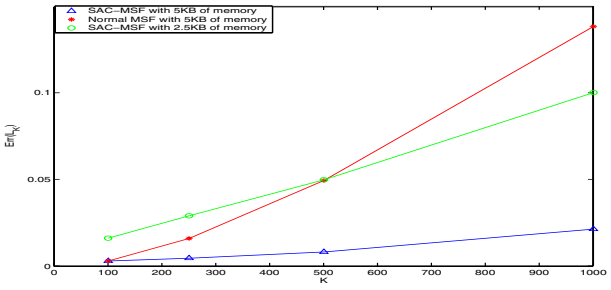

Fig. 9. FRG trace. The errors in estimation of the top K flows. MSF without any AC (with 40Kbit of memory), MSF + SAC (with 40Kbit of memory) and MSF + SAC (with 20Kbit of memory).

VII. Summary

In this paper we propose two randomized active counters algorithms, SAC and HAC, prove their unbiasedness and evaluate the variance of the corresponding random variables. While active counters are essential for a number of recently proposed data streaming and/or packet sampling algorithms [6], [2], [3], [21], they can also be beneficial for other applications like remote monitoring at heavily loaded network processors (See Section I-B).

SAC uses only SRAM but exhibits larger errors than HAC which uses augmenting DRAM for storing information on the recent history to reduce the variance. Storing full counter information in SRAM and recent history information in DRAM is a paradigm shift from the previous passive counters proposals [16], [22], [17], [18] that use DRAM for full counter information and SRAM for recent history details. Having in mind the possible high load of DRAM bandwidth caused by other applications we suggest dynamic control of the DRAM access frequency. This enhancement can be directly exploited by other SRAM/DRAM proposals that usually neglect the variability of the available DRAM bandwidth.

Finally we conclude with a list of open problems that we plan to investigate in the future.

Open problem 1. Let Λ be a data streaming algorithm that uses limited memory (given by S bytes) and active counters architecture with q-bits per counter. Pick q such that cost $C(\Lambda)$ (errors, false positives, false negatives, or something else) is minimized.

Open problem 2. Characterize available DRAM bandwidth at (heavily loaded) network processors and investigate (adaptive) hybrid SRAM/DRAM schemes under variable DRAM access frequency using rich AQM theory.

Open problem 3. The proposed solutions can be immediately extended to multi-operations arithmetics (present solutions consider only additive counters). Could the theory be extended? Are there applications that need it?

VIII. Acknowledgements

This work is supported by the Science Foundation Ireland grant 04/IN3/I460. The author would like to thank his PhD advisor Robert Shorten for continuous support as well as Steven Strachan, Ken Duffy, Dave Malone, Orla McGann, Peter Clifford and the anonymous reviewers for useful comments and suggestions.

References

[1] G. Appenzeller, I. Keslassy, N. McKeown. “Sizing router buffers”. Proc. of ACM SIGCOMM ’04, Portland, Oregon, USA, 2004.
[2] S. Cohen, Y. Matias. “Spectral bloom filters” Proc. of the ACM SIGMOD ’03, San Diego, CA, USA, 2003.
[3] N. Duffield, C. Lund, M. Thorup. “Charging from sampled network usage”. Proc. of the 1st ACM SIGCOMM Workshop on Internet Measurements, San Francisco, California, USA, 2001.
[4] N. Duffield. “Sampling for Passive Internet Measurement: A Review”. Statistical Science, Volume 19, no. 3, 2004, Pages 472-498.
[5] C. Estan. “Comparison between multistage filters and sketches for finding heavy hitters”. UCSD technical report CS2004-0784, April 2004
[6] C. Estan, G. Varghese. “New directions in traffic measurement and accounting”. ACM Transactions on Computer Systems, Vol 21 , (3), 2003.
[7] W. Feng, D. D. Kandlur, D. Saha, K. G. Shin. “A Self-Configuring RED Gateway”. Proc. of INFOCOM, New York, NY, USA, 1999.
[8] S. Floyd, R. Gummadi, S. Shenker. “Adaptive RED: An algorithm for increasing the robustness of RED’s active queue management”. August 2001, online: http://www.icir.org/floyd/papers/adaptiveRed.pdf.
[9] S. Floyd, V. Jacobson. “Random early detection gateways for congestion avoidance”. *IEEE/ACM Transactions on Networking*, vol. 1, Aug. 1993.
[10] R. J. Gibbens, F. P. Kelly. “Distributed Connection Acceptance Control for a Connectionless Network”. 16th International Teletraffic Conference, Edimburgh, June 1999, pp. 397-413.
[11] B. Krishnamurthy, S. Sen, Y. Zhang, Y. Chen. “Sketch-based change detection: methods, evaluation, and applications”. Proc. of the ACM SIGCOMM IMC. Miami Beach, Florida, USA, 2003.
[12] A. Kumar, J. Xu, L. Li, J. Wang. “Data streaming algorithms for efficient and accurate estimation of flow size distribution”. Proc. of ACM SIGMETRICS ’04, New York, NY, USA, 2004.
[13] S. Kunniyur, R. Srikant. “Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management”. *IEEE/ACM Transactions on Networking*, vol. 12, no. 2, 286-299, April 2004.
[14] NLANR IP traces, online: http://pma.nlanr.net/Special/.
[15] NLANR IP traces, online,MRA (ftp://pma.nlanr.net/traces/daily/20050416/MRA-1113621669-1.tsh.gz), FRG (http://pma.nlanr.net/Traces/Traces/old/20011015/FRG-1114614867-1.tsh.gz).
[16] S. Ramabhadran, G. Varghese. “Efficient implementation of a statistics counter architecture”. Proc. of Sigmetrics, San Diego, USA, 2003.
[17] M. Roeder, B. Lin. “Maintaining exact statistics counters with a multi-level counter memory”. Proc. of IEEE Globecom, Dalas, USA, 2004.
[18] D. Shah, S. Iyer, B. Prabhakar, N. McKeown. “Analysis of a statistics counter architecture”. IEEE Micro, 22(1):76-81, 2002.
[19] H. Song, S. Dharmapurikar, J. Turner, J. Lockwood. “Fast hash table lookup using extended bloom filter: an aid to network processing”. Proc. of SIGCOMM, Philadelphia, USA, 2005.
[20] R. Stanojevic. “Small active counters”. Hamilton Institute Technical report, 2006. Available online: http://www.hamilton.ie/person/rade/counters.pdf.
[21] Y. Zhang, S. Singh, S. Sen, N. Duffield, C. Lund. “Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications”. Proc. ACM IMC ’04. Taormina, Sicily, Italy, 2004.
[22] Q. Zhao, J. Xu, Z. Liu. “Design of a Novel Statistics Counter Architecture with Optimal Space and Time Efficiency”. Proc. of ACM SIGMETRICS ’06, France, 2006.