An Efficient Algorithm for Measuring Medium- to Large-sized Flows in Network Traffic

Ashwin Lall
Georgia Inst. of Technology

Mitsunori Ogihara
University of Miami

Jun (Jim) Xu
Georgia Inst. of Technology

Abstract—**It has been well recognized that identifying very large flows (i.e., elephants) in a network traffic stream is important for a variety of network applications ranging from traffic engineering to anomaly detection. However, we found that many of these applications have an increasing need to monitor not only the few largest flows (say top 20), but also all of the medium-sized flows (say top 20,000). Unfortunately, existing techniques for identifying elephant flows at high link speeds are not suitable and cannot be trivially extended for identifying the medium-sized flows. In this work, we propose a hybrid SRAM/DRAM algorithm for monitoring all elephant and medium-sized flows with strong accuracy guarantees. We employ a synopsis data structure (sketch) in SRAM to filter out small flows and preferentially sample medium and large flows to a flow table in DRAM. Our key contribution is to show how to maximize the use of SRAM and DRAM available to us by using a SRAM/DRAM hybrid data structure that can achieve more than an order of magnitude higher SRAM efficiency than previous methods. We design a quantization scheme that allows our algorithm to "read just enough" from the sketch at SRAM speed, without sacrificing much estimation accuracy. We provide analytical guarantees on the accuracy of the estimation and validate these by means of trace-driven evaluation using real-world packet traces.**

I. Introduction

Accurately measuring and monitoring network traffic is essential for today's network management needs. Identifying the large flows, often referred to as elephants or heavy-hitters, and tracking their size is recognized as one of the fundamental problems in traffic monitoring [9]. The very large flows, such as those carrying more than 1% of the total traffic, are clear candidates for monitoring in applications such as threshold accounting and real-time traffic monitoring, and significant research effort has been devoted to the design of efficient solutions to this problem [1], [9], [13].

However, there is no unambiguous definition of the boundary that separates the large flows that should be tracked from small flows that can be ignored. There are several applications that require keeping track of some of the medium-sized flows as well. Take for example the problem of provisioning and enforcing bandwidth usage by Quality of Service (QoS) sensitive applications. To enforce pre-negotiated sending rates, a mechanism that can identify all flows that send faster than the pre-negotiated rate is needed. Clearly such a mechanism would need to be capable of tracking all flows with rate larger than the minimum negotiable QoS guarantee. In other words, to enable fine-grained QoS provisioning, we need mechanisms that can track flow-sizes at the same fine granularity with reasonable accuracy. A variety of applications such as usage accounting and billing, traffic engineering, etc. would benefit similarly with an expansion of the range of flow sizes for which accurate identification and tracking is available.

The fundamental challenge in tracking a large number of flows is the limited availability of fast memory (SRAM) needed to support per-packet updates to a data structure containing per-flow state. Due to the large and unpredictable number of flows in real network traffic, existing solutions, such as NetFlow [16], resort to packet sampling for reducing both the number of flows being tracked and the frequency with which the corresponding data structures are updated. While sampling is effective in reducing the packet-processing load, this is often achieved at the cost of a severe degradation in accuracy. This inaccuracy comes from the inefficient allocation of memory and bandwidth under random packet sampling, as explained next.

In today's Internet it is commonly observed that a few elephant flows account for a large fraction of the traffic, but there is also a very large number of very small flows (mice). With random packet sampling, most of the samples are collected from these elephant and mice flows, and proportionally less are collected from medium size flows. As constant relative error is desirable in many network monitoring applications, random packet sampling leads to an over-allocation of resources to these elephant flows because relative estimation error increases when the flow size to be measured decreases. On the other hand, among the large number of mice flows, a fair percentage of them may be sampled into the flow table, wasting the precious SRAM space used for storing the per-flow state.

In this work, we propose a hybrid SRAM/DRAM scheme for accurately identifying large to medium sized flows and tracking their size. Our solution is designed to provide statistical guarantees that limit possible errors in estimation. In particular, we give an approximation scheme that takes as user-defined parameters the tolerable relative error and the desired success probability and estimates the size of each flow above the threshold within the tolerable error with at least that success probability. We also show that since each large flow (of size s) only gets sampled about $O(\log(s))$ times, we can keep the flow records in relatively slow DRAM. Our algorithm does not allow for any false negatives (i.e., we detect every flow above the threshold) and we give a tight bounds on the number of false positives.

978-1-4244-3513-5/09/$25.00 ©2009 IEEE

II. Background and Related Work

The problem of maintaining the counts for a large number of flows has been studied extensively in the literature [3], [4], [10], [11]. Approximate counts for heavy-hitters have also been studied [9], [13]. In network traffic, the large number of flows and the small inter-arrival times between packets make it infeasible to maintain per-flow state in an efficient manner. This effect was first observed in the NSFNET T1 backbone, and in [2] uniform sampling was proposed as a solution. We call this a *first generation* technique. Sampling is commonly used by Cisco's Netflow [16], which is the primary tool used by network operators to monitor flow-sizes. However, as pointed out in [8], NetFlow has several shortcomings, including a lack of adaptability to anomalous traffic (e.g., worm or DDoS attacks), difficulties in choosing the sampling rate, and lack of time binning for applications that require periodic summaries.

We refer to the techniques introduced by Estan and Varghese in [9] as the *second generation* techniques. The common idea in their algorithms is that the flow tables are kept in SRAM so that per-packet updates are possible. Since these algorithms were designed specifically for a small number of large flows, they could fit all their flow records in a small amount of SRAM. The challenge was then to identify the elephants correctly. To this end they introduced two techniques: *Sample and Hold* and *Multi-stage filter*. The drawback of these methods is that the number of elephants that they can effectively keep track of is bounded by the size of the SRAM.

In [11] a scheme called ACCEL-RATE (a generalization of a previous scheme called RATE [14]) is proposed to compute flow rates. It differs from our work in that it does not explicitly maintain flow counts. Also, it assumes that the distribution of flows is stationary over time and attempts to compute their rates as fast as possible whereas we make no assumption about how the rate of any flow varies with time.

Duffield et al. addressed the problem of picking which flows to keep given a limited size flow table [5], [6], [7]. The idea of using *sketch-guided* sampling to redistribute the number of samples based on flow size in an online manner was first proposed in [15]. More recently, Hu et al. [12] proposed an adaptive non-linear sampling method (ANLS), though it is not clear that their method gives any savings in memory consumption since they create flow records for every sampled flow.

III. Our Algorithm

In this paper we propose a *third generation* hybrid technique that maximally utilizes the SRAM and DRAM resources available to us. In our proposed algorithm, we move the flow table back into the DRAM so that we can fit all the medium-sized flows in the flow table. Additionally, we use the SRAM to perform the task of filtering out the smaller flows, by making use of a sketch data structure. The sketch automatically removes the burden of having to perform expensive updates for tens of millions of small flows that are typically found in network traffic. Having this sketch available also gives us the ability to leverage the additional information it provides to make our sampling mechanism more intelligent, as described next.

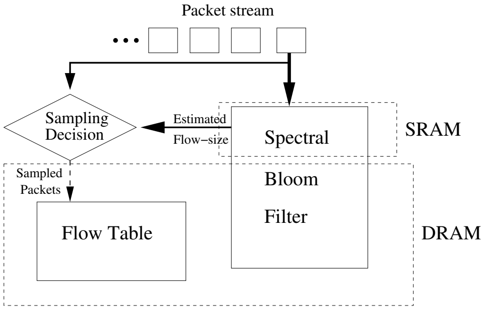


Fig. 1. Architecture of the proposed solution.

A. Algorithm

Our algorithm has two data structures—a sketch located in SRAM and the flow table located in DRAM. We make use of the sketch to sieve out the vast bulk of tiny flows that are typical in network traffic. With most of these tiny flows taken care of, our algorithm performs significantly better on the large flows. Once we verify using the sketch that a given flow is greater than our threshold, we create a record for it in the flow table and sample it from that point on. The sampling function that we use is a modification of the sketch-guided [15] one:

$$f(i) = \frac{1}{1 + \epsilon^2 \delta i},$$

and we increment the flow's count by $(1 + \epsilon^2 \delta i)$.

For the sketch, we use the spectral Bloom filter data structure to maintain approximate counts of each flow, and use this value to pick the probability with which we sample a particular packet. The spectral Bloom filter can only have over-counting error, which is exponentially small in the number of bits allocated per item inserted. We shall denote this error by δ_{Bloom}.

B. Analysis

We fix the actual count of a given flow h to be s. Let u_i denote the event that the ith occurrence of the flow is sampled in the above algorithm. First, note that the estimate returned above is unbiased with high probability (the probability that the Bloom filter is correct). The spectral filter can only fail by over-counting, and so the only way that the algorithm can start sampling a given flow at the wrong time is if the flow is over-counted before it actually reaches the threshold. If this does not happen, then the estimate is clearly unbiased since we always increment by the reciprocal of the sampling probability.

1) Quantitative Guarantee: Next, we will show that not only is our estimator unbiased (with high probability) but that it has provably small estimation error. Our guarantee is in the form of a randomized approximation, i.e., we guarantee that our estimate of the count of the flow has small relative error with high probability.

Theorem 1: For every flow larger than the threshold on which the sketch data structure does not err, our algorithm reports an (ϵ, δ)-approximation (for any pre-specified $\epsilon, \delta > 0$) of the size of the flow.

Proof: The variance for a flow of size $s > T$ can be bounded as follows:

$$\begin{aligned}\mathrm{Var}[\mathit{count}(h)] &\leq \mathrm{Var}\left[\sum_{i=1}^{s} u_i\right] = \sum_{i=1}^{s} \mathrm{Var}[u_i] \\ &= \sum_{i=1}^{s}(1/f(i) - 1) \leq s(1/f(s) - 1) \\ &= s^2\epsilon^2\delta.\end{aligned}$$

Now, we get by Chebyshev's inequality,

$$\Pr\left[|s - \mathit{count}(h)| \geq \epsilon s\right] \quad \leq \quad \frac{\mathrm{Var}[\mathit{count}(h)]}{\epsilon^2 s^2} = \delta.$$

■

Hence, for each flow h (with count $s \geq T$), we get that the estimate for it is within ϵ relative error with probability at least $1-\delta$, assuming that the Bloom filter does not err. Taking account the failure of the Bloom filter as well, our probability of error goes up to $\delta + \delta_{Bloom}$.

2) Qualitative Guarantee: Next, we analytically demonstrate that our method closely captures the medium-sized flows in the stream. First, note that as long as a flow is reported to be above the threshold by the sketch data structure, our algorithm produces a flow record for it. Hence, our algorithm only allows false positives.

To study the false positive rate, we will bound the probability with which a flow that is half the threshold will be reported as an elephant by our algorithm. The only way that we would report a flow as an elephant is if the sketch data structure seriously overcounts the flow. Recall that spectral Bloom filter works by mapping each of the m packets to k positions in an array (of size b) and increments the counters at those locations. Hence, the expected number of packets that get mapped to a given location in the array is mk/b. For a flow of size $T/2$ to incorrectly be reported an elephant, an extra $T/2$ packets would have to be mapped to all k of its positions in the array. We can use Markov's inequality to bound the probability of this happening at one location:

$$\Pr(X \geq T/2) \leq \frac{2\mathrm{E}[X]}{T} \leq \frac{2mk}{bT},$$

where X is the number of additional packets that got mapped to the same location in the array. Since the hash functions are chosen independently, the probability that all k locations have more than $T/2$ extra packets mapped to them is at most $(2mk/bT)^k$. By a similar analysis, we get the following theorem:

Theorem 2: The probability that a flow of size $T(1 - 1/d)$ has a record created for it by our algorithm (and is hence a false positive) is bounded by $(dmk/bT)^k$.

3) Number of samples: Next, we show that for each flow above the threshold we only sample it a few times.

Theorem 3: For any flow of size $s > T$, the expected number of times our algorithm samples it is at most $\frac{\ln(s/T)}{\delta\epsilon^2}$.

Proof: The expected number of samples for a flow of size $s > T$ is bounded by

$$\begin{aligned}\mathrm{E}\left[\sum_{i=T+1}^{s} u_i\right] &= \sum_{i=T+1}^{s} \mathrm{E}[u_i] = \sum_{i=T+1}^{s} f(i) \\ &= \sum_{i=T+1}^{s} \frac{1}{1+\delta\epsilon^2 i} \leq \sum_{i=T+1}^{s} \frac{1}{\delta\epsilon^2 i} \\ &\leq \frac{\ln(s/T)}{\delta\epsilon^2}.\end{aligned}$$

■

Hence, a count of size s is, in expectation, sampled only $O(\log(s/T))$ times (taking ϵ and δ to be constants). What is important to note here is that naive sampling will sample a linear number of packets in the size of the flow, whereas our method is logarithmic in the flow size.

IV. Quantization Points

In this section we discuss a modification to the sketch data structure that significantly improves the SRAM-efficiency and consequently the accuracy of our method. Note that while the counters for the spectral Bloom filter require between 32 to 64 bits [17] of SRAM, the SBF can instead use a hybrid SRAM/DRAM counter array scheme that can reduce each counter to just 4 bits [17], [18], [19]. However, such a drastic saving comes at a cost: in all these counter array schemes, the read operations can be performed only at DRAM speed. This is not acceptable to our scheme because we need to read the sketch upon every packet arrival at SRAM speed.

We propose a quantization scheme to solve this problem of "reading being too slow," which allows our scheme to use the counter architecture proposed in [19]. We use a few additional (say 4) SRAM bits per bit to indicate the approximate value range of the actual counter. Upon each packet arrival, we will read just these additional bits and make the decision whether to sample this packet. Our key insight here is that, while we increment counters in the sketch at each packet arrival, we use these counters only to check if the corresponding flow is above the threshold. In other words, while the sketch tracks flow size in a fairly precise manner, we only need coarse-grained indications about the flow size.

We now show how to choose the quantization points while still maintaining the accuracy guarantees of our sampling process. Let the quantization points be denoted by q_0, q_1, q_2, q_3, ..., q_{R-1}, assuming that we allocate $\lg R$ bits for quantization. Since we have no interest in knowing the counts of items below the threshold T, we have $q_0 = 0$ and $q_1 = T$.

For each quantization point q_i, we pick the sampling rate, p_i, in such a way that the variance in the interval between q_i and q_{i+1} is no more than that of the smooth sampling function from [15]. To do this, we always take the sampling rate to be $\frac{1}{1+\epsilon^2\delta q_i}$, i.e., the highest rate in the interval. Next, we show how to choose the remaining quantization points, $q_2, q_3, q_4, \ldots, q_{R-1}$. Note that we require $q_1 < q_2 < q_3 < \ldots < q_{R-1} = max$, where max is an upper bound on the largest flow observed.

For the quantized sampling function, the variance is:

$$\begin{aligned} V &= \sum_{i=1}^{R-2} (q_{i+1} - q_i)(1/f(q_i) - 1) \\ &= \sum_{i=1}^{R-2} (q_{i+1} - q_i)(\epsilon^2 \delta q_i). \end{aligned}$$

We would like to choose our quantization points to minimize this variance. Setting each $\frac{\partial V}{\partial q_i}$ to zero, we get the critical point for which V is extremized (it is easy to show that none of the points on the boundary work for us). We get that for each $i \geq 2$,

$$\frac{\partial V}{\partial q_i} = \epsilon^2 \delta (q_{i+1} - 2q_i + q_{i-1}).$$

Setting each of these to zero and rearranging, we get that for each $i \geq 2$, $q_{i+1} - q_i = q_i - q_{i-1}$. Hence, we should choose the quantization points such that they are equidistant from each other.

A. Analysis

We first observe that even with this quantization scheme, our estimator is still unbiased. Since the threshold T is always a quantization point, we always start sampling at the correct value. Furthermore, the estimate for each count is unbiased because, as earlier, for each sampled packet we increment the count by the inverse of the probability with which it was sampled. We now provide a sufficient condition under which our algorithm gives the same accuracy guarantee.

Theorem 4: If we pick the quantization points such that for each $1 \leq i < R - 1$, $q_{i+1} - q_i \leq T$ holds, then the variance of the estimator for each flow of size s is at most $\epsilon^2 \delta s^2$, as before.

Proof: The proof is a modification of that of Theorem 1 and is omitted here in the interest of space. ■

Hence, we can choose the quantization points in any way such that no two consecutive points are more than T apart. One way we can do this is to let the quantization points simply be $q_i = i * T$, for each $0 \leq i \leq R - 1$.

B. Number of Samples

We will now show that the number of samples under this quantization scheme for an elephant flow still only requires very few samples.

Theorem 5: Under this quantization scheme the expected number of times a flow of size s is sampled is bounded by $\frac{\log(s/T)+1}{\epsilon^2 \delta}$.

Proof: The proof of this theorem is similar to that of Theorem 3 and is omitted here in the interest of space. ■

Hence, we see that quantizing the counters of the sketch data structure allows us to have more counters (and hence less false positives), but at the same time allows us to have similar bounds on both accuracy and the number of samples of our algorithm.

TABLE I
COMPARISON FOR DESTINATION ADDRESSES IN TRACE 1

Algorithm	Recall	Avg. Rel. Err.	False pos.
Sampling	1.0000	0.1159	165
Sample and Hold	0.9486	0.1230	2246
Multi-stage Filter	0.2553	0.0005	1
Bloom-guided	1.0000	0.0385	244
Quantized	1.0000	0.0330	5

V. PERFORMANCE EVALUATION

We evaluated our two algorithms, comparing them to the current art, both qualitatively and quantitatively. In particular, we wish to measure not just what fraction of the actual elephants were detected (and how many false positives were reported), but also how accurate the estimated counts for the elephants were.

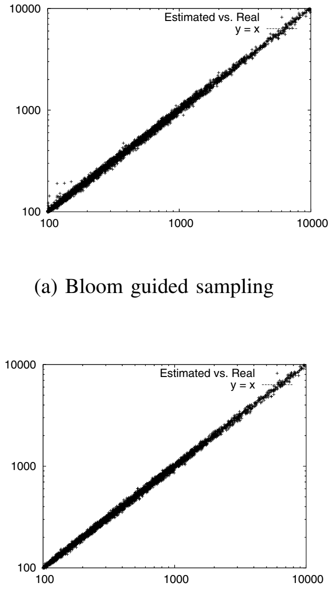
Datasets: We use two packet-header traces, binned into 10-minute epochs, for evaluating the accuracy of our algorithm. Our first packet trace (Trace 1) is an hour-long packet trace collected from USC's Los Nettos collecting facility on Feb 2, 2004 with roughly 10 million TCP packets per epoch. The second trace is an hour-long trace collected at UNC on Apr 24, 2003 with roughly 20 million packets per epoch.

Distributions of Interest: We focus on two distributions for our evaluation: the source and destination addresses of the observed network traffic. When monitoring flows to determine whether they are exceeding their quota, we would be interested in monitoring for individual IP addresses. Similarly, when trying to detect the proliferation of a worm over a network, we would like statistics on each IP address to identify the infected nodes.

Comparative Algorithms: We first evaluate the accuracy of estimation of our algorithm by comparing it against (i) a simple sampling solution, (ii) Sample and Hold [9], and (iii) Multi-stage filters [9].

In our experiments, we kept the amount of SRAM available and the maximum sampling rate to DRAM the same and compared the results of these algorithms. As in [9], we assume that a single flow table entry is equivalent to 10 counters in SRAM. In [9] there is analysis showing how to choose the sampling rate for Sample and Hold and how to distribute the SRAM into the flow table and sketch for Multi-stage Filter, but we found that these parameters did not always give optimal results. Hence, we ran these algorithms with varying parameters, using the results from the optimal ones. By doing so we show that we can outperform these two algorithms irrespective of the parameters used by them.

We focus on three metrics to evaluate the algorithms: recall, average relative error, and false positives. Recall is defined as the number of target flows correctly found above the threshold divided by the actual number above the threshold. We define relative error as the absolute error divided by the *actual* count. Finally, false positives is the count of flows that appear to be above the threshold when they really are not. All experiments were done over five independent runs and the standard deviations were very small, so we do not report them for clarity of presentation.



(a) Bloom guided sampling

(b) Quantized algorithm

Fig. 2. Estimated flow size vs. real flow size for medium-size flows

We present the results of our simulations on the destination IP addresses of Trace 1 in Table I. All the algorithms are reasonably good at identifying all the flows above the threshold, except for the Multi-stage Filter. The Multi-stage Filter misses most of the flows of interest because the flow table in SRAM is smaller since it must share its space with the filter. In comparison, the other algorithms identify almost all the flows above the threshold, with our Bloom-guided and quantized algorithms finding every single one.

Our algorithms outperform Sample and Hold with respect to average relative error specifically because of the accuracy with which we identify the medium-sized flows. The error in Sample and Hold comes entirely from how many packets are missed before the flow is sampled, and in the case of medium-sized flows this can be quite large relative to the flow size.

In Figure 2 we give the plots of the real counts versus the estimate counts of medium-sized flows for both our algorithms. They have all their estimates clustered in a band along the $y = x$ line. What is important to observe here is that our algorithms consistently give the same relative error, irrespective of the flow size.

Lastly, we take a look at the number of false positives for each algorithm (see Table I). Observe that since Sample and Hold has a high sampling rate to be sure to capture as many flows as it can, it allows in many false positives. In contrast, the Multi-stage Filter allows few to no false positives due to its sketch data structure. Our basic Bloom-guided algorithm allows in quite a few false positives, but with the quantization optimization this number is severely reduced. The reason for this is that with the quantization scheme we increase the number of counters in the spectral Bloom filter, which in turn causes far fewer collisions and hence fewer false positives. We see an improvement of upto an order of magnitude in the number of false positives because of the quantization scheme.

VI. Conclusion

In this paper we introduce and motivate the need to monitor not just the very largest flows in network traffic, but also the medium-sized ones. We show why current methods for tracking large flows are either inaccurate in estimating the flow sizes correctly or are incapable of maintaining records for all the flows of interest. Our algorithm, which is carefully designed with the constraints on the size of SRAM and speed of DRAM in mind, overcomes both these obstacles and efficiently solves the problem. We demonstrate how our algorithm needs to be adapted so as to satisfy arbitrary resource constraints placed by the hardware available.

VII. Acknowledgments

The authors would like to thank Abhishek Kumar and Erin Kozaczuk for help in editing earlier versions of this paper.

References

[1] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *ICALP*, 2002.

[2] K. C. Claffy, G. C. Polyzos, and H.-W. Braun. Application of sampling methodologies to network traffic characterization. In *SIGCOMM*, 1993.

[3] S. Cohen and Y. Matias. Spectral bloom filters. In *SIGMOD*, 2003.

[4] G. Cormode and S. M. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *Journal of Algorithms*, 55(1), 2005.

[5] N. Duffield and C. Lund. Predicting resource usage and estimation accuracy in an ip flow measurement collection infrastructure. In *IMC*, 2003.

[6] N. Duffield, C. Lund, and M. Thorup. Charging from sampled network usage. In *IMW*, 2001.

[7] N. Duffield, C. Lund, and M. Thorup. Flow sampling under hard resource constraints. In *SIGMETRICS*, 2004.

[8] C. Estan, K. Keys, D. Moore, and G. Varghese. Building a better netflow. *SIGCOMM Comput. Commun. Rev.*, 34(4):245–256, 2004.

[9] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *SIGCOMM*, 2002.

[10] P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science: Special Issue on External Memory Algorithms and Visualization*, A, 1999.

[11] F. Hao, M. Kodialam, and T. V. Lakshman. ACCEL-RATE: a faster mechanism for memory efficient per-flow traffic estimation. In *SIGMETRICS*, 2004.

[12] C. Hu, S. Wang, J. Tian, B. Liu, Y. Cheng, and Y. Chen. Accurate and efficient traffic monioring using adaptive non-linear sampling method. In *IEEE Infocom*, 2008.

[13] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems*, 28(1), 2003.

[14] M. Kodialam, T. Lakshman, , and S. Mohanty. Runs based traffic estimator (rate): A simple, memory efficient scheme for per-flow rate estimation. In *INFOCOM*, 2004.

[15] A. Kumar and J. Xu. Sketch guided sampling – using on-line estimates of flow size for adaptive data collection. In *Infocom*, 2006.

[16] Cisco Netflow. http://www.cisco.com/web/go/netflow.

[17] S. Ramabhadran and G. Varghese. Efficient implementation of a statistics counterarchitecture. In *SIGMETRICS*, 2003.

[18] D. Shah, S. Iyer, B. Prabhakar, and N. McKeown. Maintaining statistics counters in router line cards. In *IEEE Micro*, 2002.

[19] Q. Zhao, J. J. Xu, and Z. Liu. Design of a novel statistics counter architecture with optimal space and time efficiency. In *SIGMETRICS*, 2006.