This paper was presented as part of the main technical program at IEEE INFOCOM 2011

Fast and Compact Per-Flow Traffic Measurement through Randomized Counter Sharing

Tao Li† Shigang Chen† Yibei Ling‡
† Department of Computer & Information Science & Engineering, University of Florida, Gainesville, FL, USA
‡ Applied Research Laboratories, Telcordia Technologies, NJ, USA

***Abstract*—Traffic measurement provides critical real-world data for service providers and network administrators to perform capacity planning, accounting and billing, anomaly detection, and service provision. One of the greatest challenges in designing an online measurement module is to minimize the per-packet processing time in order to keep up with the line speed of the modern routers. To meet this challenge, we should minimize the number of memory accesses per packet and implement the measurement module in the on-die SRAM. The small size of SRAM requires extremely compact data structures to be designed for storing per-flow information. The best existing work, called *counter braids*, requires more than 4 bits per flow and performs 6 or more memory accesses per packet. In this paper, we design a fast and compact measurement function that estimates the sizes of all flows. It achieves the optimal processing speed: 2 memory accesses per packet. In addition, it provides reasonable measurement accuracy in a tight space where the counter braids no longer work. Our design is based on a new data encoding/decoding scheme, called *randomized counter sharing*. This scheme allows us to mix per-flow information together in storage for compactness and, at the decoding time, separate the information of each flow through statistical removal of the error introduced during information mixing from other flows. The effectiveness of our online per-flow measurement approach is analyzed and confirmed through extensive experiments based on real network traffic traces.**

I. Introduction

A. Motivation

Traffic monitoring and measurement provide critical information for capacity planning, accounting and billing, anomaly detection, and service provision in modern computer networks [1], [2], [3], [4], [5]. This paper focuses on a particularly challenging problem, the measurement of per-flow information for a high-speed link without using per-flow data structures. The goal is to estimate the *size* of each flow (in terms of number of packets). A flow is identified by a label that can be a source address, a destination address, or any combination of addresses, ports, and other fields in the packet header.

Most prior work investigates how to identify the elephants (i.e., flows with large sizes) [1], [2], [6], [7] or estimate the flow size distribution (i.e., the number of flows that have a certain size) [8], [9]. They do not measure the sizes of individual flows, which are important information for many applications. For example, if we use the addresses of the users as flow labels, per-flow traffic measurement provides the basis for usage-based billing and graceful service differentiation, where a user's service priority gracefully drops as he overspends his resource quota. Studying per-flow data over consecutive measurement periods may help us discover network access patterns and, together with user profiling, reveal geographic/demographic traffic distributions among users. Such information will help Internet service providers and application developers to align network resource allocation with the majority's needs. In the event of a worm attack, per-source data can be used to estimate the scanning rates of worm-infected hosts. In the event of a botnet attack where there is a sudden surge of small flows, a security administrator may analyze the change in the flow size distribution and use per-flow information to compile the list of candidate bots that contribute to the change, helping to narrow down the scope for further investigation.

B. Challenges and Prior Art

Per-flow traffic measurement is a very challenging problem. In order to monitor small flows with a few packets, it is desirable to record information for each packet because aggressive sampling, such as what's used in Cisco Netflow and others [10], [11], will not only introduce significant measurement error but also miss some small flows altogether. DRAM cannot keep up with today's line speed (40 Gbps for OC-768 and 16.4 Tbps in experimental systems [12]). The recent research trend is to implement online measurement functions in high-speed but expensive on-die SRAM [5], [9], [13], [14].

However, the size of SRAM is limited, and it has to be shared among many critical functions for routing, scheduling, traffic management, and security. Even for traffic measurement alone, there can be multiple functions running concurrently on the same SRAM, each measuring a different type of flows. These functions may have to take turns to access the SRAM, and the amount of memory allocated to each measurement function is likely to be small. Hence, it is extremely important to make per-flow measurement and other functions in SRAM fast and compact. In particular, to keep up with the router's forwarding rate (which can be as high as a couple of cycles per packet), we should minimize the number of memory accesses per packet performed by each measurement function to the shared SRAM.

It has been shown in [2] that maintaining per-flow counters cannot scale for high-speed links. Even for efficient counter implementations [15], [16], [17], SRAM will only be able to hold a small fraction of per-flow state (including counters and indexing data structures such as pointers and flow identities for locating the counters). The *counter braids* avoid per-flow counters and achieve near-optimal memory efficiency [18], [19]. This method maps each flow to three arbitrary counters;

978-1-4244-9921-2/11/$26.00 ©2011 IEEE

they are all incremented by one for every packet of the flow. Many flows may be mapped to the same counter, which stores the sum of the flow sizes. Essentially, the counters represent linear equations, which can be solved for the flow sizes. Two levels of counters are used to reduce the memory overhead. The counter braids require slightly more than 4 bits per flow and are able to count the exact sizes of all flows. But it also has two limitations. First, it performs 6 or occasionally 12 memory accesses per packet. Second, when the memory allocated to a measurement function is far less than 4 bits per flow, our experiments show that the message passing decoding algorithm of counter braids cannot converge to any meaningful results. When the available memory is just $1\sim2$ bits per flow, the *exact measurement* of the flow sizes is no longer possible. We have to resort to *estimation methods*. The key is to efficiently utilize the limited space to improve the accuracy of the estimated flow sizes, and do so with the minimum number of memory accesses per packet. This is what our paper tries to achieve.

A related thread of research is to collect statistical information of the flows [8], [9], or identify the largest flows and devote the available memory to measure their sizes while ignoring the smaller ones [2], [6], [7], [20]. For example, RATE [21] and ACCEL-RATE [22] measure per-flow rate by maintaining per-flow state, but they use a *two-run sampling* method to filter out small-rate flows so that only high-rate flows are measured.

Also related is the work [14] that measures the number of *distinct* destinations that each source has contacted. Per-flow counters cannot be used to solve this problem because they cannot remove duplicate packets. If a source sends 1,000 packets to a destination, the packets contribute only one contact, but will count as 1,000 when we measure the flow size. To remove duplicates, bitmaps (instead of counters) should be used [5], [23], [24], [25]. From the technical point of view, this represents a separate line of research, which employs a different set of data structures and analytical tools. Attempt has also been made to use bitmaps for estimating the flow sizes, which is however far less efficient than counters, as our experiments will show.

C. Our Contributions

We design a fast and compact per-flow traffic measurement function that achieves three main objectives: i) It shares counters among flows to save space, and does not incur any space overhead for mapping flows to their counters. This distinguishes our work from [15], [16], [17]. ii) It updates exactly one counter per packet, which is optimal. This separates our work from the counter braids that update three or more counters per packet. Updating each counter requires two memory accesses for read and then write. iii) It provides estimation of the flow sizes, as well as the confidence intervals that characterize the accuracy, even when the available memory is too small such that other exact-counting methods including [18], [19] no longer work. We believe our work is the first one that achieves all these objectives. It complements the existing work by providing additional flexibility for the practitioners to choose when other methods cannot meet the speed and space requirements.

The design of our measurement function is based on a new data encoding/decoding scheme, called *randomized counter sharing*. It splits the size of each flow among a number of counters that are randomly selected from a counter pool. These counters form the *storage vector* of the flow. For each packet of a flow, we randomly select a counter from the flow's storage vector and increment the counter by one. Such a simple online operation can be implemented very efficiently. The storage vectors of different flows share counters uniformly at random; the size information of one flow in a counter is the noise to other flows that share the same counter. Fortunately, this noise can be quantitatively measured and removed through statistical methods, which allow us to estimate the size of a flow from the information in its storage vector. We propose two estimation methods whose accuracies are statistically guaranteed. They work well even when the total number of counters in the pool is by far smaller than the total number of flows that share the counters. Our experimental results based on real traffic traces demonstrate that the new methods can achieve good accuracy in a tight space.

The randomized counter sharing scheme proposed in this paper for per-flow traffic measurement has applications beyond the networking field. It may be used in the data streaming applications to collect per-item information from a stream of data items.

II. Performance Metrics

We measure the number of packets in each flow during a measurement period, which ends every time after a certain number (e.g., 10 millions) of packets are processed. The design of per-flow measurement functions should consider the following three key performance metrics.

A. Processing Time

The per-packet processing time of an online measurement function determines the maximum packet throughput that the function can operate at. It should be made as small as possible in order to keep up with the line speed. This is especially true when multiple routing, security, measurement, and resource management functions share SRAM and processing circuits.

The processing time is mainly determined by the number of memory accesses and the number of hash computations (which can be efficiently implemented in hardware [26]). The counter braids [18], [19] update three counters at the first level for each packet. When a counter at the first level overflows, it needs to update three additional counters at the second level. Hence, it requires 3 hashes and 6 memory accesses to read and then write back after counter increment. But in the worse case, it requires 6 hashes and 12 memory accesses. The multi-resolution space-code Bloom filters [13] probabilistically select one or more of its 9 filters and set $3\sim6$ bits in each of the selected ones. Each of those bits requires one memory access and one hash computation.

Our objective is to achieve a constant per-packet processing time of one hash computation and two memory accesses (for updating a single counter). This is the minimum processing

time for any method that use hash operations to identify counters for update.

B. Storage Overhead

The need to reduce the SRAM overhead has been discussed in the introduction section. One may argue that because the amount of memory needed is related to the number of packets in a measurement period, we can reduce the memory requirement by shortening the measurement period. However, when the measurement period is smaller, more flows will span multiple periods and consequently the average flow size in each period will be smaller. When we measure the flow sizes, we also need to capture the flow labels [19], e.g., a tuple of source address/port and destination address/port to identify a TCP flow. The flow labels are too large to fit in SRAM. They have to be stored in DRAM. Therefore, in a measurement period, each flow incurs at least one DRAM access to store its flow label. If the average flow size is large enough, the overhead of this DRAM access will be amortized over many packets of a flow. However, if the average flow size is too small, the DRAM access will become the performance bottleneck that seriously limits the throughput of the measurement function. This means the measurement period should not be too small. Our experiments in Section VIII set a measurement period such that the average flow size is about 10.

C. Estimation Accuracy

Let s be the size of a flow and $\hat{s}$ be the estimated size of the flow based on a measurement function. The estimation accuracy of the function can be specified by a confidence interval: the probability for s to be within $[\hat{s}\cdot(1-\beta), \hat{s}\cdot(1+\beta)]$ is at least a pre-specified value α, e.g., 95%. A smaller value of β means that the estimated flow size is more accurate (in a probabilistic sense).

There is a tradeoff between the estimation accuracy and the storage overhead. If the storage space and the processing time are unrestricted, we can accurately count each packet to achieve perfect accuracy. However, in practice, there will be constraints on both storage and processing speed, which make 100% accurate measurement sometimes infeasible. In this case, one has to settle with imperfect results that can be produced with the available resources. Within the bounds of the limited resources, we must explore novel measurement methods to make the estimated flow sizes as accurate as possible.

III. System Design

A. Basic Idea

We use an example to illustrate the idea behind our new measurement approach. Suppose the amount of SRAM allocated to one of the measurement functions is 2Mb (2×2^{20} bits), and each measurement period ends after 10 million packets, which translate into about 8 seconds for an OC-192 link (10+ Gbps) with an average packet size of 1,000 bytes. The types of flows that the online functions may measure include per-source flows, per-destination flows, per-source/destination flows, TCP flows, WWW flows (with destination port 80), etc. Without losing generality, suppose the specific function under consideration in this example measures the sizes of TCP flows.

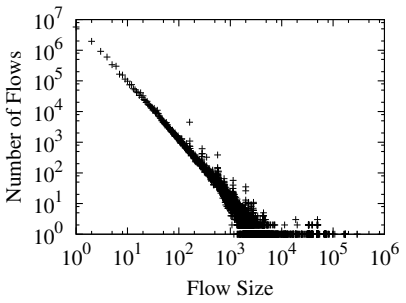

Fig. 1. Traffic distribution: each point shows the number (y coordinate) of flows that have a certain size (x coordinate).

Fig. 1 shows the number of TCP flows that have a certain flow size in log scale, based on a real network trace captured by the main gateway of our campus. If we use 10 bits for each counter, there will be only 0.2 million counters. The number of concurrent flows in our trace for a typical measurement period is around 1 million. It is obvious that allocating per-flow state is not possible and each counter has to store the information of multiple flows. But if an "elephant" flow is mapped to a counter, that counter will overflow and lose information. On the other hand, if only a couple of "mouse" flows are mapped to a counter, the counter will be under-utilized, with most of its high-order bits left as zeros.

To solve the above problems, we not only store the information of multiple flows in each counter, but also store the information of each flow in a large number of counters, such that an "elephant" is broken into many "mice" that are stored at different counters. More specifically, we map each flow to a set of l randomly-selected counters and split the flow size into l roughly-equal shares, each of which is stored in one counter. The value of a counter is the sum of the shares from all flows that are mapped to the counter. Because flows share counters, they introduce noise to each other's measurement. The key to accurately estimate the size of a flow is to measure the noise introduced by other flows in the counters that the flow is mapped to.

Fortunately, this can be done if the flows are mapped to the counters uniformly at random. Any two flows will have the same probability of sharing counters, which means that each flow will have the same probability of introducing a certain amount of noise to any other flow. If the number of flows and the number of counters are very large, the combined noise introduced by all flows will be distributed across the counter space about uniformly. The statistically uniform distribution of the noise can be measured and removed. The above scheme of information storage and recovery is called *randomized counter sharing*.

We stress that this design philosophy of "splitting" each flow among a large number of counters is very different from "replicating" each flow in three counters as the counting Bloom filter [27] or counter braids [18] do — they add the size of each flow as a whole to three randomly selected counters. Most notably, our method increments one counter for each

arrival packet, while the counting Bloom filter or counter braids increment three counters. We store the information of each flow in many counters (e.g., 50), while they store the information of each flow in three counters.

B. Overall Design

Our online traffic measurement function consists of two modules. The online data encoding module stores the information of arrival packets in an array of counters. For each packet, it performs one hash function to identify a counter and then updates the counter with two memory accesses, one for reading and the other for writing. At the end of each measurement period, the counter array is stored to the disk and then reset to zeros.

The offline data decoding module answers queries for flow sizes. It is performed by a designated offline computer. We propose two methods for separating the information about the size of a flow from the noise in the counters. The first one is called the *counter sum estimation method* (CSM), which is very simple and easy to compute. The second one is called the *maximum likelihood estimation method* (MLM), which is more accurate but also more computationally intensive. The two complementary methods provide flexibility in designing a practical system, which may first use CSM for rough estimations and then apply MLM to the ones of interest.

The problem of collecting flow labels has technically been treated as a separate problem in the literature [18]. It can be efficiently solved [19] and thus will not be further addressed in this paper.

IV. Online Data Encoding

The flow size information is stored in an array C of m counters. The ith counter in the array is denoted as $C[i]$, $0 \leq i \leq m-1$. The size of the counters should be set so that the chance of overflow is negligible; we will discuss this issue in details in Section VII. Each flow is mapped to l counters that are randomly selected from C through hash functions. These counters logically form a *storage vector* of the flow, denoted as C_f, where f is the label of the flow. The ith counter of the vector, denoted as $C_f[i]$, $0 \leq i \leq l-1$, is selected from C as follows:

$$C_f[i] = C[H_i(f)], \tag{1}$$

where $H_i(...)$ is a hash function whose range is $[0, m)$. We want to stress that C_f is **not** a separate array for flow f. It is merely a logical construction from counters in C for the purpose of simplifying the presentation. In all our formulas, one should treat the notation $C_f[i]$ simply as $C[H_i(f)]$. The hash function H_i, $0 \leq i \leq l-1$, can be implemented from a master function $H(...)$ as follows: $H_i(f) = H(f|i)$ or $H_i(f) = H(f \oplus R[i])$, where '|' is the concatenation operator, '$\oplus$' is the XOR operator, and $R[i]$ is a constant whose bits differ randomly for different indices i.

All counters are initialized to zeros at the beginning of each measurement period. The operation of online data encoding is very simple: When the router receives a packet, it extracts the flow label f from the packet header, randomly selects a counter from C_f, and increases the counter by one. More specifically, the router randomly picks a number i between 0 and $l-1$, computes the hash $H_i(f)$, and increases the counter $C[H_i(f)]$, which is physically in the array C, but logically the ith element in the vector C_f.

V. Offline Counter Sum Estimation

A. Estimation Method

At the end of a measurement period, the router stores the counter array C to a disk for long-term storage and offline data analysis. Let n be the combined size of all flows, which is $\sum_{i=0}^{m-1} C[i]$. Let s be the true size of a flow f during the measurement period. The estimated size, $\hat{s}$, based on our counter sum estimation method (CSM) is

$$\hat{s} = \sum_{i=0}^{l-1} C_f[i] - l\frac{n}{m}. \tag{2}$$

The first item is the sum of the counters in the storage vector of flow f. It can also be interpreted as the sum of the flow size s and the noise from other flows due to counter sharing. The second item captures the expected noise. Below we formally derive (2).

Consider an arbitrary counter in the storage vector of flow f. We treat the value of the counter as a random variable X. Let Y be the portion of X contributed by the packets of flow f, and Z be the portion of X contributed by the packets of other flows. Obviously, $X = Y + Z$.

Each of the s packets in flow f has a probability of $\frac{1}{l}$ to increase the value of the counter by one. Hence, Y follows a binomial distribution:

$$Y \sim Bino(s, \frac{1}{l}). \tag{3}$$

Each packet of another flow f' has a probability of $\frac{1}{m}$ to increase the counter by one. That is because the probability for the counter to belong to the storage vector of flow f' is $\frac{l}{m}$, and if that happens, the counter has a probability of $\frac{1}{l}$ to be selected for increment. Assume there is a large number of flows, the size of each flow is negligible when comparing with the total size of all flows, and l is large such that each flow's size is randomly spread among many counters. We can approximately treat the packets independently. Hence, Z approximately follows a binomial distribution:

$$Z \sim Bino(n-s, \frac{1}{m}) \approx Bino(n, \frac{1}{m}), \text{ because } s \ll n. \tag{4}$$

We must have

$$E(X) = E(Y+Z) = E(Y) + E(Z) = \frac{s}{l} + \frac{n}{m}. \tag{5}$$

That is,

$$s = l \times E(X) - l\frac{n}{m}. \tag{6}$$

From the observed counter values $C_f[i]$, $E(X)$ can be measured as $\frac{\sum_{i=0}^{l-1} C_f[i]}{l}$. We have the following estimation for s:

$$\hat{s} = \sum_{i=0}^{l-1} C_f[i] - l\frac{n}{m}. \tag{7}$$

If a flow shares a counter with an "elephant" flow, its size estimation can be skewed. However, our experiments show that CSM works well in general because the number of "elephants" is typically small (as shown in Fig. 1) and thus their impact is also small, particularly when there are a very large number of counters and flows. Moreover, our next method based on maximum likelihood estimation can effectively reduce the impact of an outlier in a flow's storage vector that is caused by an "elephant" flow. In addition, the estimated result $\hat{s}$ can be negative for some flow. In this case, we manually set it to 1.

B. Estimation Accuracy

The mean and variance of $\hat{s}$ will be given in (9) and (10), respectively. They are derived as follows: Because $X = Y+Z$, we have

$$\begin{aligned} E(X^2) &= E((Y+Z)^2) = E(Y^2) + 2E(YZ) + E(Z^2) \\ &= E(Y^2) + 2E(Y)E(Z) + E(Z^2) \\ &= \frac{s^2}{l^2} - \frac{s}{l^2} + \frac{s}{l} + 2 \cdot \frac{s}{l} \cdot \frac{n}{m} + \frac{n^2}{m^2} - \frac{n}{m^2} + \frac{n}{m}. \end{aligned}$$

The following facts are used in the above mathematical process: $E(Y^2) = \frac{s^2}{l^2} - \frac{s}{l^2} + \frac{s}{l}$ because $Y \sim Bino(s, 1/l)$. $E(YZ) = E(Y)E(Z)$ since Y and Z are independent. $E(Z^2) = \frac{n^2}{m^2} - \frac{n}{m^2} + \frac{n}{m}$ because $Z \sim Bino(n, 1/m)$.

$$\begin{aligned} Var(X) &= E(X^2) - (E(X))^2 \\ &= \frac{s}{l}(1 - \frac{1}{l}) + \frac{n}{m}(1 - \frac{1}{m}). \end{aligned} \tag{8}$$

In (7), $C_f[i]$, $0 \le i \le l-1$, are independent samples of X. We can interpret $\hat{s}$ as a random variable in the sense that a different set of samples of X may result in a different value of $\hat{s}$. From (7), we have

$$\begin{aligned} E(\hat{s}) &= l \times E(X) - l\frac{n}{m} \\ &= l(\frac{s}{l} + \frac{n}{m}) - l\frac{n}{m} \quad = s, \end{aligned} \tag{9}$$

which means our estimation is unbiased. The variance of $\hat{s}$ can be written as

$$\begin{aligned} Var(\hat{s}) &= l^2 \times Var(X) = l^2\left(\frac{s}{l}(1 - \frac{1}{l}) + \frac{n}{m}(1 - \frac{1}{m})\right) \\ &= s(l-1) + l^2\frac{n}{m}(1 - \frac{1}{m}). \end{aligned} \tag{10}$$

C. Confidence Interval

The confidence interval for the estimation will be given in (13), and it is derived as follows: The binomial distribution, $Z \sim Bino(n, 1/m)$, can be closely approximated as a Gaussian distribution, $Norm(\frac{n}{m}, \frac{n}{m}(1-\frac{1}{m}))$, when n is large. Similarly, the binomial distribution, $Y \sim Bino(s, \frac{1}{l})$, can be approximated by $Norm(\frac{s}{l}, \frac{s}{l}(1 - \frac{1}{l}))$. Because the linear combination of two independent Gaussian random variables is also normally distributed [28], we have $X \sim Norm(\frac{s}{l} + \frac{n}{m}, \frac{s}{l}(1 - \frac{1}{l}) + \frac{n}{m}(1 - \frac{1}{m}))$. To simplify the presentation, let $\mu = \frac{s}{l} + \frac{n}{m}$ and $\Delta = \frac{s}{l}(1 - \frac{1}{l}) + \frac{n}{m}(1 - \frac{1}{m})$.

$$X \sim Norm(\mu, \Delta), \tag{11}$$

where the mean μ and the variance Δ agree with (5) and (8), respectively.

Because $\hat{s}$ is a linear function of $C_f[i]$, $0 \le i \le l-1$, which are independent samples of X, $\hat{s}$ must also approximately follow a Gaussian distribution. From (7) and (11), we have

$$\hat{s} \sim Norm(s, s(l-1) + l^2\frac{n}{m}(1 - \frac{1}{m})). \tag{12}$$

Hence, the confidence interval is

$$\hat{s} \pm Z_\alpha\sqrt{s(l-1) + l^2\frac{n}{m}(1 - \frac{1}{m})}, \tag{13}$$

where α is the confidence level and Z_α is the α percentile for the standard Gaussian distribution. As an example, when $\alpha = 95\%$, $Z_\alpha = 1.96$.

VI. Maximum Likelihood Estimation

In this section, we propose the second estimation method that is more accurate but also more computationally expensive.

A. Estimation Method

We know from the previous section that any counter in the storage vector of flow f can be represented by a random variable X, which is the sum of Y and Z, where $Y \sim Bino(s, \frac{1}{l})$ and $Z \sim Bino(n, 1/m)$. For any integer $z \in [0, n)$, the probability for the event $Z = z$ to occur can be computed as follows:

$$Pr\{Z = z\} = \binom{n}{z}(\frac{1}{m})^z(1 - \frac{1}{m})^{n-z}.$$

Because n and m are known, $Pr\{Z = z\}$ is a function of a single variable z and thus denoted as $P(z)$.

Based on the probability distribution of Y and Z, the probability for the observed value of a counter, $C_f[i]$, $\forall i \in [0, l)$, to occur is

$$\begin{aligned} &Pr\{X = C_f[i]\} \\ &= \sum_{z=0}^{C_f[i]} (Pr\{Z = z\} \cdot Pr\{Y = C_f[i] - z\}) \\ &= \sum_{z=0}^{C_f[i]} \binom{s}{C_f[i] - z}(\frac{1}{l})^{C_f[i]-z}(1 - \frac{1}{l})^{s-(C_f[i]-z)}P(z). \end{aligned} \tag{14}$$

Let $y = C_f[i] - z$ to simplify the formula. The probability for all observed values in the storage vector of flow f to occur is

$$\begin{aligned} L &= \prod_{i=0}^{l-1} Pr\{X = C_f[i]\} \\ &= \prod_{i=0}^{l-1}\left(\sum_{z=0}^{C_f[i]} \binom{s}{y}(\frac{1}{l})^y(1 - \frac{1}{l})^{s-y}P(z)\right). \end{aligned} \tag{15}$$

The maximum likelihood method (MLM) is to find an estimated size $\hat{s}$ of flow f that maximizes the above likelihood function. Namely, we want to find

$$\hat{s} = \arg\max_s\{L\}. \tag{16}$$

To find $\hat{s}$, we first apply logarithm to turn the right side of the equation from product to summation.

$$\ln(L) = \sum_{i=0}^{l-1} \ln\left(\sum_{z=0}^{C_f[i]} \binom{s}{y} (\frac{1}{l})^y (1-\frac{1}{l})^{s-y} P(z) \right). \quad (17)$$

Because $\frac{d\binom{s}{y}}{ds} = \binom{s}{y}(\psi(s+1) - \psi(s+1-y))$, where $\psi(...)$ is the polygamma function [29], we have

$$\frac{d(\binom{s}{y}(1-\frac{1}{l})^{s-y})}{ds} = \binom{s}{y}(1-\frac{1}{l})^{s-y}\left(\psi(s+1) - \psi(s+1-y) + \ln(1-\frac{1}{l})\right).$$

To simplify the presentation, we denote the right side of the above equation as $O(s)$. From (17), we can compute the first-order derivative of $\ln(L)$ as follows:

$$\frac{d\ln(L)}{ds} = \sum_{i=0}^{l-1} \frac{\sum_{z=0}^{C_f[i]} \left(O(s)(\frac{1}{l})^y P(z) \right)}{\sum_{z=0}^{C_f[i]} \binom{s}{y} (\frac{1}{l})^y (1-\frac{1}{l})^{s-y} P(z)}. \quad (18)$$

Maximizing L is equivalent to maximizing $\ln(L)$. Hence, by setting the right side of (18) to zero, we can find the value for $\hat{s}$ through numerical methods. Because $\frac{d\ln(L)}{ds}$ is a monotone function of s, we have used the bisection search method in all our experiments in Section VIII to find the value $\hat{s}$ that makes $\frac{d\ln(L)}{ds}$ equal to zero.

B. Estimation Accuracy

The estimation confidence interval will be given in (26), and it is derived as follows: The estimation formula is given in (16). According to the classical theory for MLM [30], when l is sufficiently large, the distribution of the flow-size estimation $\hat{s}$ can be approximated by

$$Norm(s, \frac{1}{\mathcal{I}(\hat{s})}), \quad (19)$$

where the *fisher information* $\mathcal{I}(\hat{s})$ [31] of L is defined as follows:

$$\mathcal{I}(\hat{s}) = -E\left(\frac{d^2\ln(L)}{ds^2}\right). \quad (20)$$

In order to compute the second-order derivative, we begin from (11) and have the following:

$$Pr\{X = C_f[i]\} = \frac{1}{\sqrt{2\pi\Delta}} e^{-\frac{(C_f[i]-\mu)^2}{2\Delta}}$$
$$\ln(Pr\{X = C_f[i]\}) = -\ln(\sqrt{2\pi\Delta}) - \frac{(C_f[i]-\mu)^2}{2\Delta}, \quad (21)$$

where $0 \le i \le l-1$. Performing the second-order differentiation, we have

$$\frac{d^2\ln(Pr\{X = C_f[i]\})}{ds^2} = -\frac{\mu'}{l\Delta} + \frac{(\frac{1}{2}(1-\frac{1}{l}) + \mu - C_f[i])\Delta'}{l\Delta^2} + \frac{1}{l\Delta^3}(1-\frac{1}{l})\left((\mu - C_f[i])\mu'\Delta - (\mu - C_f[i])^2\Delta'\right), \quad (22)$$

where $\mu' = \frac{1}{l}$ and $\Delta' = \frac{1}{l}(1-\frac{1}{l})$. Therefore,

$$E(\frac{d^2\ln(Pr\{X = C_f[i]\})}{ds^2}) = -\frac{\mu'}{l\Delta} + \frac{\frac{1}{2}(1-\frac{1}{l})\Delta'}{l\Delta^2} + \frac{1}{l\Delta^3}(1-\frac{1}{l})E(\mu - C_f[i])^2\Delta' = -\frac{1}{l^2\Delta} + \frac{3(1-\frac{1}{l})^2}{2l^2\Delta^2}, \quad (23)$$

where we have used the following facts: $E(\mu - C_f[i]) = 0$ and $E(\mu - C_f[i])^2 = \Delta$. Because $L = \prod_{i=0}^{l-1} Pr\{X = C_f[i]\}$, we have

$$\mathcal{I}(\hat{s}) = -E\left(\frac{d^2\ln(L)}{ds^2}\right) = \sum_{i=0}^{l-1} E(\frac{d^2\ln(Pr\{X = C_f[i]\})}{ds^2}) = -\frac{1}{l^2\Delta} + \frac{3(1-\frac{1}{l})^2}{2l^2\Delta^2}. \quad (24)$$

From (19), the variance of $\hat{s}$ is

$$Var(\hat{s}) = \frac{1}{\mathcal{I}(\hat{s})} = \frac{2l^2\Delta^2}{3(1-\frac{1}{l})^2 - 2\Delta}. \quad (25)$$

Hence, the confidence interval is

$$\hat{s} \pm Z_\alpha \cdot \sqrt{\frac{2l^2\Delta^2}{3(1-\frac{1}{l})^2 - 2\Delta}}, \quad (26)$$

where Z_α is the α percentile for the standard Gaussian distribution.

VII. Setting Counter Length and Handling Overflow Problem

So far, our analysis has assumed that each counter has a sufficient number of bits such that it will not overflow. However, in order to save space, we want to set the counter length as short as possible. Suppose each measurement period ends after a pre-specified number n of packets are received. (Note that the value of n is the combined sizes of all flows during each measurement period.) The average value of all counters will be $\frac{n}{m}$. We set the number of bits in each counter, denoted as b, to be $\log_2 \frac{n}{m} + 1$. Due to the additional bit, each counter can hold at least two times of the average before overflowing. If the allocated memory has M bits, the values of b and m can be determined from the following equations:

$$b \times m = M, \qquad \log_2 \frac{n}{m} + 1 = b. \quad (27)$$

Due to the randomized counter sharing design, roughly speaking, the packets are distributed in the counters at random. We observe in our experiments that the counter values approximately follow a Gaussian distribution with a mean of $\frac{n}{m}$. In this distribution, the fraction of counters that are more than four times of the mean is very small — less than 5.3% in all our experiments. Consequently, the impact of counter overflow in CSM or MLM is also very small for most flows.

To totally eliminate the impact of counter overflow, one approach is to keep another array of counters in DRAM, each of which has a sufficient number of bits. The counters in DRAM are one-to-one mapped to the counters in SRAM. When a counter in SRAM is overflowed, it is reset to zero and

TABLE I
NUMBER OF MEMORY ACCESSES AND NUMBER OF HASH COMPUTATIONS PER PACKET

	memory accesses	hash computations	constant?
CSM	2	1	Y
MLM	2	1	Y
CB	≥ 6	≥ 3	N
MRSCBF	4.47	4.47	N

the corresponding counter in DRAM is incremented by one. During offline data analysis, the counter values are set based on both SRAM and DRAM data. Because overflow happens only to a small fraction of SRAM counters and a DRAM access is made only after an overflowed SRAM counter is accessed 2^b times, the overall overhead of DRAM access is very small.

VIII. EXPERIMENTS

We use experiments to evaluate our estimation methods, CSM (Counter Sum estimation Method) and MLM (Maximum Likelihood estimation Method), which are designed based on the randomized counter sharing scheme. We also compare our methods with CB (Counter Braids) [18] and MRSCBF (Multi-Resolution Space-Code Bloom Filters) [13]. Our evaluation is based on the performance metrics outlined in Section II, including per-packet processing time, memory overhead, and estimation accuracy.

The experiments use a network traffic trace obtained from the main gateway of our campus. We perform experiments on various different types of flows, such as per-source flows, per-destination flows, per-source/destination flows, and TCP flows. They all lead to the same conclusions. Without losing generality, we choose TCP flows for presentation. The trace contains about 68 millions of TCP flows and 750 millions of packets. In each measurement period, 10 million packets are processed; it typically covers slightly more than 1 million flows.

A. Processing Time

The processing time is mainly determined by the number of memory accesses and the number of hash computations per packet. Table I presents the comparison. CSM or MLM performs two memory accesses and one hash computation for each packet. CB incurs three times of the overhead. It performs six memory accesses and three hash computations for each packet at the first counter level, and in the worst case makes six additional memory accesses and three additional hash computations at the second level. MRSCBF has nine filters. The ith filter uses k_i hash functions and encodes packets with a sampling probability p_i, where $k_1 = 3$, $k_2 = 4$, $k_i = 6$, $\forall i \in [3, 9]$, and $p_i = (\frac{1}{4})^{i-1}$, $\forall i \in [1, 9]$. When encoding a packet, the ith filter performs k_i hash computations and sets k_i bits. Hence, the total number of memory accesses (or hash computations) per packet for all filters is $\sum_{i=1}^{9}(p_i \cdot k_i) \approx 4.47$.

B. Memory Overhead and Estimation Accuracy

We study the estimation accuracies of CSM and MLM under different levels of memory availability. In each measurement period, 10M packets are processed, i.e., $n = 10$M, which translates into about 8 seconds for an OC-192 link (10+ Gbps) or about 2 seconds for an OC-768 link (40+ Gbps) with an average packet size of 1,000 bytes. The memory M allocated to this particular measurement function is varied from 2Mb (2×2^{20} bits) to 8Mb. The counter length b and the number of counters m are determined based on (27). The size of each storage vector is 50.

When $M = 2$Mb, the experimental results are presented in Fig. 2. The first plot from the left shows the estimation results by CSM for one measurement period; the results for other measurement periods are very similar. Each flow is represented by a point in the plot, whose x coordinate is the true flow size s and y coordinate is the estimated flow size $\hat{s}$. The equality line, $y = x$, is also shown for reference. An estimation is more accurate if the point is closer to the equality line.

The second plot presents the 95% confidence intervals for the estimations made by CSM. The width of each vertical bar shows the size of the confidence interval at a certain flow size (which is the x coordinate of the bar). The middle point of each bar shows the mean estimation for all flows of that size. Intuitively, the estimation is more accurate if the confidence interval is smaller and the middle point is closer to the equality line.

The third plot shows the estimation results by MLM, and the fourth plot shows the 95% confidence intervals for the estimations made by MLM. Clearly, MLM achieves better accuracy than CSM. The estimation accuracy shown in Fig. 2 is achieved with a memory of slightly less than 2 bits per flow,

We can improve the estimation accuracy of CSM or MLM by using more memory. We increase M to 4Mb and repeat the above experiments. The results are shown in Fig. 3. We then increase M to 8Mb and repeat the above experiments. The results are shown in Fig. 4. The accuracy clearly improves as the confidence intervals shrink when M becomes larger.

We repeat the same experiments on CB, whose parameters are selected according to [18]. The results are presented in Fig. 5. The first plot shows that CB totally fails to produce any meaningful results when the available memory is too small: $M = 2$Mb, which translates into less than 2 bits per flow. In fact, its algorithm cannot converge, but instead produce oscillating results. We have to artificially stop the algorithm after a very long time. The second plot shows that CB works well when $M = 4$Mb. The algorithm still cannot converge by itself, even though it can produce very good results when we artificially stop it after a long time without observing any further improvement in the results. It can be seen that the results carry a small positive bias because most points are on one side of the equality line. The third plot shows that CB is able to return the exact sizes for most flows when the memory is $M = 8$Mb.

Combining the results in Table I, we draw the following

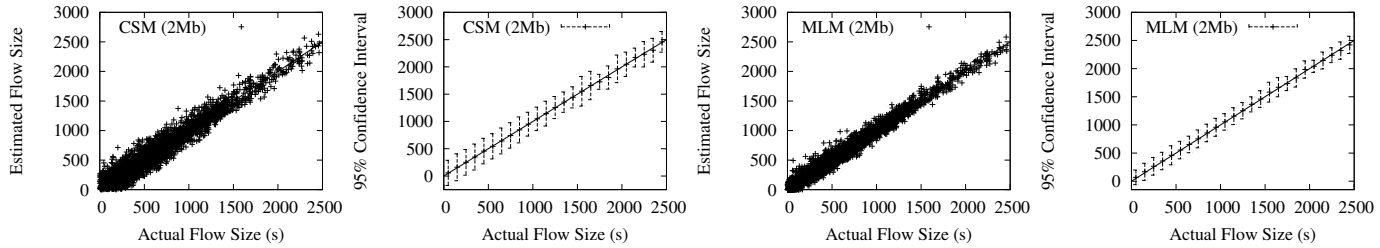

Fig. 2. • *First Plot*: estimation results by CSM when $M = 2$Mb. • *Second Plot*: 95% confidence intervals for the estimations made by CSM when $M = 2$Mb. • *Third Plot*: estimation results by MLM when $M = 2$Mb. • *Fourth Plot*: 95% confidence intervals for the estimations made by MLM when $M = 2$Mb.

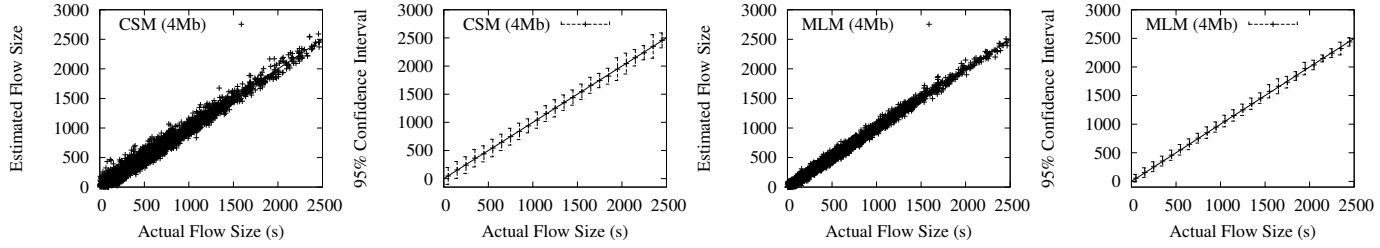

Fig. 3. • *First Plot*: estimation results by CSM when $M = 4$Mb. • *Second Plot*: 95% confidence intervals for the estimations made by CSM when $M = 4$Mb. • *Third Plot*: estimation results by MLM when $M = 4$Mb. • *Fourth Plot*: 95% confidence intervals for the estimations made by MLM when $M = 4$Mb.

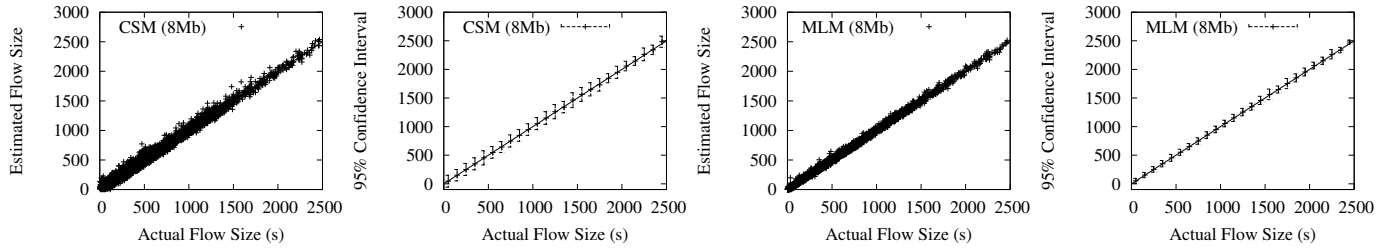

Fig. 4. • *First Plot*: estimation results by CSM when $M = 8$Mb. • *Second Plot*: 95% confidence intervals for the estimations made by CSM when $M = 8$Mb. • *Third Plot*: estimation results by MLM when $M = 8$Mb. • *Fourth Plot*: 95% confidence intervals for the estimations made by MLM when $M = 8$Mb.

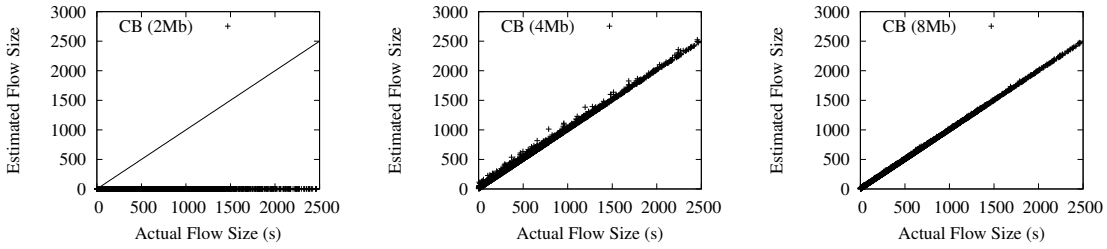

Fig. 5. • *First Plot*: estimation results by CB when $M = 2$Mb. • *Second Plot*: estimation results by CB when $M = 4$Mb. • *Third Plot*: estimation results by CB when $M = 8$Mb.

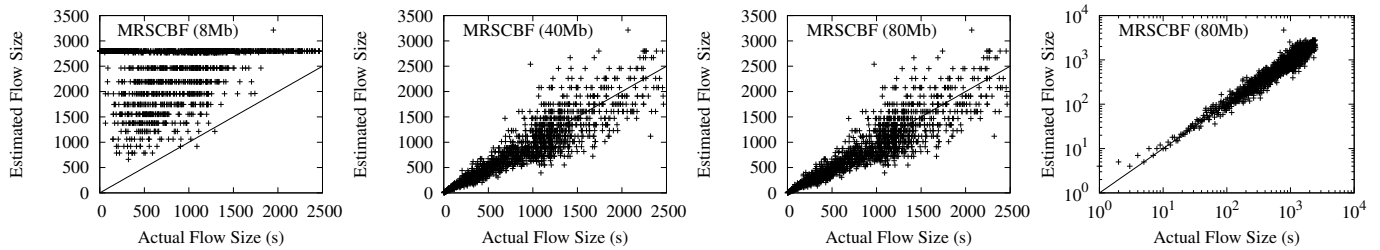

Fig. 6. • *First Plot*: estimation results by MRSCBF when $M = 8$Mb. • *Second Plot*: estimation results by MRSCBF when $M = 40$Mb. • *Third Plot*: estimation results by MRSCBF when $M = 80$Mb. • *Fourth Plot*: estimation results in logarithmic scale by MRSCBF when $M = 80$Mb.

conclusion: (1) In practice, we should choose CSM/MLM if the requirement is to handle high measurement throughput (which means low per-packet processing time) or if the available memory is too small such that CB does not work, while relatively coarse estimation is acceptable. (2) We should choose CB if the processing time is less of a concern, sufficient memory is available, and the exact flow sizes are required.

We also run MRSCBF under different levels of memory availability. We begin with $M = 8$Mb. CSM or MLM works very well with this memory size (Fig. 4). The performance of MRSCBF is shown in the first plot of Fig. 6. There are some very large estimated sizes. To control the scale in the vertical axis, we artificially set any estimation beyond 2,800 to be 2,800. The results demonstrate that MRSCBF totally fails when $M = 8$Mb. The performance of MRSCBF improves when we increase the memory. The results when $M = 40$Mb are shown in the second plot.[1] In the third plot, when we further increase M to 80Mb,[2] no obvious improvement is observed when comparing the second plot. A final note is that the original paper of MRSCBF uses log scale in their presentation. The third plot in Fig. 6 will appear as the fourth plot in log scale.

Clearly, the bitmap-based MRSCBF performs worse than CB, CSM or MLM. To measure flow sizes, counters are superior than bitmaps.

IX. Conclusion

Per-flow traffic measurement provides real-world data for a variety of applications on accounting and billing, anomaly detection, and traffic engineering. Current online data collection methods cannot meet the requirements of being both fast and compact. This paper proposes a novel data encoding/decoding scheme, which mixes per-flow information randomly in a tight SRAM space for compactness. Its online operation only incurs a small overhead of one hash computation and one counter update per packet. Two offline statistical methods — the counter sum estimation and the maximum likelihood estimation — are used to extract per-flow sizes from the mixed data structures with good accuracy. Due to its fundamentally different design philosophy, the new measurement function is able to work in a tight space where exact measurement is no longer possible, and it does so with the minimal number of memory accesses per packet.

References

[1] X. Dimitropoulos, P. Hurley, and A. Kind, "Probabilistic Lossy Counting: An Efficient Algorithm for Finding Heavy Hitters," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 1, pp. 7–16, 2008.

[2] C. Estan and G. Varghese, "New Directions in Traffic Measurement and Accounting," *Proc. of ACM SIGCOMM*, August 2002.

[3] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-Based Change Detection: Methods, Evaluation, and Applications," *Proc. of ACM SIGCOMM Internet Measurement Conference*, October 2003.

[1] At the end of each measurement period, about half of the bits in the filters of MRSCBF are set to ones.

[2] At the end of each measurement period, less than half of the bits in the filters of MRSCBF are set to ones.

[4] X. Li, F. Bian, M. Crovella, C. Diot, R. Govindan, G. Iannaccone, and A. Lakhina, "Detection and Identification of Network Anomalies Using Sketch Subspaces," *Proc. of ACM SIGCOMM Internet Measurement Conference*, October 2006.

[5] M. Yoon, T. Li, S. Chen, and J. Peir, "Fit a Spread Estimator in Small Memory," *Proc. of IEEE INFOCOM*, April 2009.

[6] N. Kamiyama and T. Mori, "Simple and Accurate Identification of High-rate Flows by Packet Sampling," *Proc. of IEEE INFOCOM*, April 2006.

[7] R. Karp, S. Shenker, and C. Papadimitriou, "A Simple Algorithm for Finding Frequent Elements in Streams and Bags," *ACM Transactions on Database Systems*, vol. 28, no. 1, pp. 51–55, 2003.

[8] N. Duffield, C. Lund, and M. Thorup, "Estimating Flow Distributions from Sampled Flow Statistics," *Proc. of ACM SIGCOMM*, October 2003.

[9] A. Kumar, M. Sung, J. Xu, and J. Wang, "Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Size Distribution," *Proc. of ACM SIGMETRICS*, June 2004.

[10] N. Duffield, C. Lund, and M. Thorup, "Flow Sampling under Hard Resource Constraints," *Proc. of ACM SIGMETRICS/Performance*, June 2004.

[11] N. Duffield, C. Lund, and M. Thorup, "Learn More, Sample Less: Control of Volume and Variance in Network Measurement," *IEEE Transactions of Information Theory*, vol. 51, no. 5, pp. 1756–1775, 2005.

[12] W. David Gardner, "Researchers Transmit Optical Data At 16.4 Tbps," *InformationWeek*, February 2008.

[13] A. Kumar, J. Xu, J. Wang, O. Spatschek, and L. Li, "Space-Code Bloom Filter for Efficient Per-Flow Traffic Measurement," *Proc. of IEEE INFOCOM*, March 2004, A journal version was published in IEEE JSAC, 24(12):2327-2339, December 2006.

[14] S. Venkatataman, D. Song, P. Gibbons, and A. Blum, "New Streaming Algorithms for Fast Detection of Superspreaders," *Proc. of NDSS*, February 2005.

[15] D. Shah, S. Iyer, B. Prabhakar, and N. McKeown, "Maintaining Statistics Counters in Router Line Cards," *Proc. of IEEE Micro*, vol. 22, no. 1, pp. 76–81, 2002.

[16] S. Ramabhadran and G. Varghese, "Efficient Implementation of a Statistics Counter Architecture," *Proc. ACM SIGMETRICS*, June 2003.

[17] Q. Zhao, J. Xu, and Z. Liu, "Design of a Novel Statistics Counter Architecture with Optimal Space and Time Efficiency," *Proc. of ACM Sigmetrics/Performance*, 2006.

[18] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter Braids: A Novel Counter Architecture for Per-Flow Measurement," *Proc. of ACM SIGMETRICS*, June 2008.

[19] Y. Lu and B. Prabhakar, "Robust Counting Via Counter Braids: An Error-Resilient Network Measurement Architecture," *Proc. of IEEE INFOCOM*, April 2009.

[20] E. Demaine, A. Lopez-Ortiz, and J. Munro, "Frequency Estimation of Internet Packet Streams with Limited Space," *Proc. of 10th ESA Annual European Symposium on Algorithms*, September 2002.

[21] M. Kodialam, T. V. Lakshman, and S. Mohanty, "Runs bAsed Traffic Estimator (RATE): A Simple, Memory Efficient Scheme for Per-Flow Rate Estimation," *Proc. of INFOCOM*, March 2004.

[22] F. Hao, M. Kodialam, and T. V. Lakshman, "ACCEL-RATE: A Faster Mechanism for Memory Efficient Per-flow Traffic Estimation," *Proc. of ACM SIGMETRICS/Performance*, June 2004.

[23] C. Estan, G. Varghese, and M. Fish, "Bitmap Algorithms for Counting Active Flows on High-Speed Links," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. 5, pp. 925–937, October 2006.

[24] Q. Zhao, A. Kumar, and J. Xu, "Joint Data Streaming and Sampling Techniques for Detection of Super Sources and Destinations," *Proc. of USENIX/ACM Internet Measurement Conference*, October 2005.

[25] J. Cao, Y. Jin, A. Chen, T. Bu, and Z. Zhang, "Identifying High Cardinality Internet Hosts," *Proc. of IEEE INFOCOM*, April 2009.

[26] M. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient Hardware Hashing Functions for High Performance Computers," *IEEE Transactions on Computers*, vol. 46, no. 12, pp. 1378–1381, 1997.

[27] S. Cohen and Y. Matias, "Spectral Bloom Filters," *Proc. of ACM SIGMOD*, June 2003.

[28] G. Casella and R. Berger, "Statistical Inference," *Duxbury Press*, June 2001.

[29] M. Abramowitz and I. Stegun, "Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables," *Dover Publications*, June 1964.

[30] Myung Jay I. and Navarro Daniel J., "Information Matrix," *http://faculty.psy.ohio-state.edu/myung/personal/info_matrix.pdf*, 2004.

[31] E. Lehmann and G. Casella, "Theory of Point Estimation," *Springer Press*, 1998.