Space-Code Bloom Filter for Efficient Traffic Flow Measurement

Abhishek Kumar   Jun (Jim) Xu
College of Computing
Georgia Institute of Technology
{akumar,jx}@cc.gatech.edu

Li Li
Bell Labs
Lucent
erranlli@lucent.com

Jia Wang
AT&T Labs - Research
jiawang@research.att.com

***Abstract*— Per-flow traffic measurement is critical for usage accounting, traffic engineering, and anomaly detection. Previous methodologies are either based on random sampling (e.g., Cisco's NetFlow), which is inaccurate, or only account for the "elephants". Our paper introduces a novel technique for measuring per-flow traffic approximately, for all flows regardless of their sizes, at very high-speed (say, OC192+). The core of this technique is a novel data structure called Space Code Bloom Filter (SCBF). A SCBF is an approximate representation of a multiset; each element in this multiset is a traffic flow and its multiplicity is the number of packets in the flow. SCBF employs a Maximum Likelihood Estimation (MLE) method to measure the multiplicity of an element in the multiset. Through parameter tuning, SCBF allows for graceful tradeoff between measurement accuracy and computational and storage complexity. SCBF also contributes to the foundation of data streaming by introducing a new paradigm called blind streaming. We evaluated the performance of SCBF on packet traces gathered from a tier-1 ISP backbone and through mathematical analysis. Our preliminary results demonstrate that SCBF achieves reasonable measurement accuracy with very low storage and computational complexity.**

I. Introduction

Accurate traffic measurement and monitoring is critical for network management. For example, per-flow traffic accounting has applications in usage-based charging/pricing, security, per-flow QoS, and traffic engineering [1]. While there has been considerable research on characterizing the statistical distribution of per-flow traffic [2] or on identifying and measuring a few large flows (elephants) [3], [1], [4], little work has been done on investigating highly efficient algorithms and data structures to facilitate per-flow measurement on very high-speed links.

To fill this gap, we propose a novel data structure called Space-Code Bloom Filter (SCBF) and explore its applications to network measurement in general, and to per-flow traffic accounting in particular. A (traditional) bloom filter [5] is an approximate representation of a set S, which given an arbitrary element x, allows for the membership query "$x \in S$?". A Space-Code Bloom Filter (SCBF), on the other hand, is an approximate representation of a multiset M, which allows for the query "how many occurrences of x are there in M?". Just as a bloom filter achieves a nice tradeoff between space efficiency (bits per element) and false positive ratio, SCBF achieves a nice tradeoff between the accuracy of counting and the number of bits used for counting.

SCBF has several important applications in network measurement. This paper focuses on its application to performing "per-flow" traffic accounting without per flow state on a high-speed link. Given a flow identifier, SCBF returns the estimated number of packets in the flow during a *measurement epoch*. Here, a flow identifier can be an IP address, a source and destination IP address pair, the combination of IP addresses and port numbers, or other indices that can identify a flow.

Per-flow accounting is a challenging task on high-speed network links. While keeping per-flow state would make accounting straightforward, it is not desirable since such a large state will only fit on DRAM and the DRAM speed can not keep up with the rate of a high-speed link. While random sampling, such as used in Cisco Netflow, reduces the requirement on memory speed, it introduces excessive measurement errors for flows other than elephants, as shown in Section II. The scheme by Estan and Varghese [1] only needs a small amount of fast memory. However, it allows the monitoring of only a small number of elephants.

Previous attempts at using bloom filters to answer multiset queries have produced a number of variations of *counting bloom filter* [6]. In its most basic form, a counting bloom filter has a counter associated with each bit in the array. When an element x is inserted in a counting bloom filter with k hash functions $h_1, \cdots, h_k$, each of the k counters associated with the bits $h_1(x), \cdots, h_k(x)$ are incremented by one. Unfortunately, quantitative estimates based on counters might be a long way off the correct value of the frequency of occurrence of any element in counting bloom filters. Approaches like *conservative update* [1] have been proposed to counter this problem to some extent. Such heuristics fail to provide any bounds on the estimation error and do not yield to analysis. Counting bloom filters are not suitable from the implementation perspective either. They require a large number of counters, each of them capable of counting up to the largest possible multiplicity, thus wasting both space and computation cycles. Attempts to improve the space efficiency of counting bloom filters have resulted in the proposal of variable size counters [7]. Unfortunately, the mechanism required to implement variable size counters is complex, and cannot match the rate of a high speed link.

Our approach is to perform traffic accounting on a very small amount of high-speed SRAM, organized as an *SCBF page*. Once an SCBF page becomes full (we formalize this notion later), it is eventually paged to persistent storages such as disks. Later, to find out the traffic volume of a flow identified by a label x during a measurement epoch, the SCBF pages corresponding to the epoch can be queried using x to provide the approximate answer. The challenges facing this approach are threefold. First, the amount of persistent storage to store SCBF pages cannot be unreasonably large, even for a high-speed link like OC-192+ (10+ Gbps). Second, the computational complexity of processing each packet needs to be low enough to catch up with the link speed. Third, the accounting needs to be fairly accurate for all the flows, despite the aforementioned storage and complexity constraints.

SCBF is designed to meet all these challenges. Our design can easily scale to maintaining approximate per-flow counts at an OC-192+ link using a limited amount of fast memory. The storage cost for a full-speed OC-192 link is tolerable: about 2 bits per packet or 9 GB per hour. Such a cost is manageable for Tier-1 ISPs as the storage cost right now is about 1 dollar per GB. In addition, it is very amenable to pipelined hardware implementation to facilitate high-speed processing.

Here we describe the conceptual design of SCBF, deferring its detailed description to Section III. An SCBF is essentially a large number of statistical estimators running in parallel. Each estimator tracks the traffic volume of a certain flow. SCBF nicely codes and compresses the current “readings” of these estimators within a small memory module so that they do not interfere with each other. Like space-time coding allows signals to multiplex on both space and time domains, SCBF allows “signals” to multiplex on both space and code domains, hence the name *Space-Code*. The demultiplexing operation for obtaining the “reading” of a given flow in an SCBF employs a Maximum Likelihood Estimation (MLE) process. We show through careful analysis that the “readings” of all flows will be accurate to a certain ratio with high probability.

SCBF not only has important applications in network measurement, but also contributes to the foundation of *data streaming* [8], [4]. Data streaming is concerned with processing a long stream of data items in one pass using a small working memory in order to answer a class of queries regarding the stream. The challenge is to use this small memory to “remember” as much information *pertinent to the queries* as possible. The contributions of SCBF to data streaming are twofold. First, it is among the earliest work in the networking context [4]. Although data streaming has emerged as a major field in database [8], [7], [9], the techniques invented in the database context generally cannot be “ported” to networking because they are much more expensive in computational complexity. Second, SCBF introduces a new paradigm called *blind streaming* in which incrementing the reading of an estimator does not require the decoding of its current reading, and hence the *blindness*. This significantly reduces the computational and hardware implementation complexity of each operation, as discussed in Section II-B.

The rest of this paper is organized as follows. In the next section, we revisit the motivation of this work and identify objectives and constraints specific to per-flow measurement on high-speed links. Section III describes the design of SCBF. We provide some mathematical details of SCBF in Section IV. Section V presents a preliminary evaluation over a number of large packet header traces from a tier-1 ISP IP backbone network. We conclude in Section VI with pointers to future work.

II. Architecture, Performance Metrics, and Blind Streaming

The proposed SCBF scheme is motivated by the need to provide per-flow traffic accounting at very high speed (e.g, OC192+). A naive solution to this problem would be to maintain per-flow counters that are updated upon every packet arrival. However, as shown in [1], this approach cannot scale to the link speed of OC192+ since fast SRAM modules can only hold a tiny fraction of per-flow state due to their size limitations, yet large DRAM modules cannot support such speed. Random sampling with a small rate such as 1% may make the speed requirement for keeping the per-flow state affordable for DRAM. However, they lead to intolerable inaccuracies in network measurement [1]. In particular, sampling will typically miss the majority of small flows (containing only a few packets). Ignoring these mice altogether may lead to wrong conclusions in applications such as estimation of flow distribution and network anomaly detection.

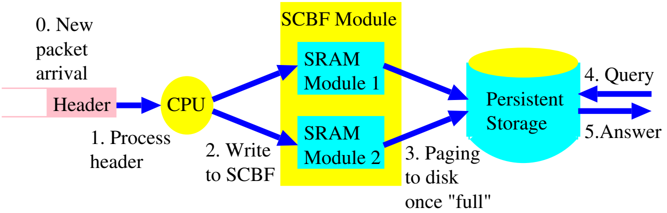

Fig. 1. The system model for using SCBF for traffic measurement.

Our vision is to design a synopsis data structure that keeps approximate track of the number of packets in each flow regardless of its size, yet is small enough to fit in fast SRAM. The proposed SCBF scheme is a brainchild of this vision. The overall architecture of using SCBF to perform per-flow accounting is shown in Figure 1. SCBF is updated upon each packet arrival (arcs 1 and 2 in Figure 1) so that it will not fail to record the presence of any flow, small or large. When the SCBF becomes full, it will be paged to persistent storage devices (arc 3). Typically, two “ping-pong” SCBF modules will be used so that one can process new packets while the other is being paged, as shown in Figure 1. In other words, these two SCBF modules store approximate flow accounting information in alternating measurement epochs. In addition, SCBF succinctly represents a large number of counters so that paging is infrequent enough to fit within the disk bandwidth even for OC-192+ link speed. Finally, a query concerning the size of a flow can be made to a SCBF page stored on the disk (arc 4). The result of the query (arc 5) is the approximate number of packets in the flow during the measurement epoch an SCBF page records.

A. Performance Metrics

The key challenge of designing SCBF is to achieve a nice tradeoff between the following three key performance metrics.

1. Storage complexity. This refers to the amount of space consumed on persistent storage to store the SCBF pages. This can be equivalently characterized as the traffic rate between the SCBF module and the disk. Our goal is to make this complexity as small as possible, given a fixed link speed. At least this rate should not exceed the disk bandwidth. We will show that this complexity is manageable even on OC192+ speed since SCBF takes advantage of the "Quasi-Zipf Law" of the Internet traffic: a small number of flows contribute to the majority of Internet traffic and the majority of flows are small.

2. Computational complexity. We are also concerned with the number of memory accesses to the SCBF module for each packet. This has to be minimized. We show that our scheme will incur no more than 6 bits of write per packet to the memory. We will see later that most of these writes overwrite bits that are already 1, thus filling up the SCBF page at a much slower rate.

3. Accuracy of estimation. We would like our estimation of the traffic volume in a measurement epoch to be as close to the actual value as possible. In this paper, our goal is *constant relative error tolerance*, i.e., for the estimate $\hat{F}$ to be within $[(1-\epsilon)F, (1+\epsilon)F]$ with high probability. Here F is the actual value. This is achieved using a maximum likelihood estimator (MLE).

Clearly, very high accuracy can be achieved if one is willing to spend more storage and computational complexity. Therefore, there is an inherent tradeoff between the complexities and the accuracy. This tradeoff is exploited through a sophisticated parameter tuning process, which will be detailed in Section V.

B. Blind Streaming

A careful reader may notice that in Figure 1, we do not have an arc from the SCBF module to the CPU. She may also wonder whether this is a mistake, since when a new packet arrives, its flow identifier should be used to look up a corresponding entry for update. In fact, our SCBF is designed to avoid such a read before update, i.e., the SCBF data structure is write-only! We refer to this feature as *blind streaming*, in the sense that reading and decoding data in the SCBF is not required before updating it.

Blind streaming is a new paradigm of data streaming that is especially suitable for high-speed networks for the following reasons. First, in blind streaming, we do not need to deal with the race condition between read and write operations, making a pipelined hardware implementation extremely simple. Note that in traditional data processing, a datum has to be locked after read and unlocked after write to ensure consistency. Second, blind streaming also doubles the streaming speed by eliminating the reading process. Third, the loss of accuracy due to this blindness is tolerable, as we will show in Section V.

III. Design Details

A. Space-Code Bloom Filter

The core of our scheme is a novel data structure called Space-Code Bloom Filter (SCBF). It approximately represents a multiset, extending the capability of a traditional Bloom Filter (BF) to represent a set. Given an element x, it not only allows one to check if x is in a multiset, but also counts the number of occurrences of x. In the following, we describe the design of both BF and SCBF.

A traditional bloom filter representing a set $S = \{x_1, x_2, .., x_n\}$ of size n is described by an array A of m bits,

Insertion phase (given x):
    $i = rand(1, l)$;
    Set bits $A[h_1^i(x)]$, ..., $A[h_k^i(x)]$ to 1;

Query phase (given y):
    $\hat{\theta} = 0$;
    for($i = 1$; $i \leq l$; $i++$)
        if (bits $A[h_1^i(x)]$, ..., $A[h_k^i(x)]$ are all 1)
            $\hat{\theta} = \hat{\theta} + 1$;
    return MLE($\hat{\theta}$);

Fig. 2. Insertion and Query in SCBF

initialized to 0. A Bloom filter uses k independent hash functions $h_1, h_2, ..., h_k$ with range $\{1, ..., m\}$. We refer to this set of hash functions as a *group*. In the *insertion phase*, given an element x to be inserted into a set S, we set the bits $A[h_i(x)]$, $1 \leq i \leq k$, to 1. In the *query phase*, to check if an element y is in S, we check the value of the bits $A[h_i(y)]$, $i = 1, 2, ..., k$. The answer to the query is *yes* if all these bits are 1, and *no* otherwise.

A bloom filter guarantees not to have any false negatives, i.e., returning "no" while the set contains the element. However, it may contain false positives, i.e., returning "yes" while the element is not in the set. There is a convenient tradeoff between the false positive and the number of elements the filter tries to hold. It was shown in [5] that fixing a false positive threshold γ, the filter can hold the highest number of elements n when the parameter k is set to around $(-\log_2 \gamma)$. In this case the completely full filter contains exactly half 1's and half 0's. We refer to this as the "50% golden rule" in the sequel.

The insertion and query algorithms of SCBF are shown in Figure 2. In a traditional bloom filter, once an element x is inserted, later insertions of x will write to the same bits $A[h_1(x)], A[h_2(x)], ..., A[h_k(x)]$, and will not result in any change to A. SCBF, on the other hand, uses a filter made up of l groups of hash functions $\{h_1^1(x), h_2^1(x), ..., h_k^1(x)\}$, $\{h_1^2(x), h_2^2(x), ..., h_k^2(x)\}$, ..., and $\{h_1^l(x), h_2^l(x), ..., h_k^l(x)\}$. Each group can be viewed as a traditional bloom filter. In the insertion phase, one group of hash functions $\{h_1^i(x), h_2^i(x), ..., h_k^i(x)\}$ is chosen randomly, and the bits $A[h_1^i(x)], A[h_2^i(x)], ..., A[h_k^i(x)]$ are set to 1. In the query phase, to find out the number of occurrences of element y in the set, we count the number of groups that y has matched. An element y matches a group $\{h_1^i, h_2^i, ..., h_k^i\}$ if the bits $A[h_1^i(y)], A[h_2^i(y)], ..., A[h_k^i(y)]$ are all 1. Based on the number of groups that y has matched, denoted as $\hat{\theta}$, we use a maximum likelihood estimation (MLE) procedure to estimate its multiplicity in the SCBF, returning $MLE(\hat{\theta})$. We can precompute an MLE table for all possible values of $\hat{\theta}$ so that later decoding involves only a straightforward lookup. However, the theory behind the computation of the MLE table is involved and will be discussed in Section IV. We refer to the scheme as Space-Code Bloom Filter because each group can be viewed as a code for an element, and in SCBF, multiple groups spread codes of an element to a larger space.

```
Insertion phase (given x):
    for(j = 1; j <= r; j++)
        Insert x into SCBF j with probability p_j
        /*shown in Figure 2.*/

Query phase (given y):
    Check y's occurrences in SCBF 1, 2, ..., r
        and obtain counts θ̂_1, θ̂_2, ..., θ̂_r respectively;
    return MLE(θ̂_1, θ̂_2, ..., θ̂_r);
```

Fig. 3. Insertion and Query Algorithms in MRSCBF

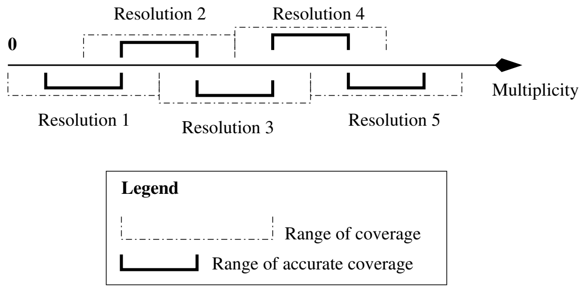

Fig. 4. The conceptual design of MRSCBF

B. Multi-Resolution Space-Code Bloom Filter

The potential count of an element can be very high. By the famous coupon collector problem, all l groups in a SCBF will be matched at least once after about $(l \ln l)$ copies of x are inserted. Accurate estimation of the number of occurrences of x will not be possible beyond this threshold. Making l very large does not solve this problem for two reasons. First, the number of false positives (noise) become large with larger l, and if the multiplicity of an element y (signal) is small, the noise will overwhelm the signal. Second, the storage efficiency of the scheme becomes low as multiple occurrences of an element are spread to a very large space.

Our solution to this problem is Multi-Resolution SCBF (MRSCBF). It employs multiple SCBFs, operating at different resolutions. Its insertion and query algorithms are shown in Figure 3. The insertion algorithm for MRSCBF is a simple extension of that of SCBF. When a packet arrives, it will result in an insertion into SCBF i with a sampling probability p_i. Suppose there are a total of r filters. Without loss of generality, we assume $p_1 > p_2 > ... > p_r$. The higher p_i value corresponds to higher resolution. Our goal is that elements with low multiplicities will be estimated by filter(s) of higher resolutions, while elements with high multiplicities will be estimated by filters of lower resolutions. In the query algorithm, we count the number of groups that x matches in filters $1, 2, ..., r$, denoted as $\hat{\theta}_1, \hat{\theta}_2, ..., \hat{\theta}_r$ respectively. The final estimate will be $MLE(\hat{\theta}_1, \hat{\theta}_2, ..., \hat{\theta}_r)$, the result of a joint MLE procedure based on the observations. Like in SCBF, the decoding table for this MLE procedure again will be precomputed. However, without any approximation, its precomputation would take years. We developed techniques,discussed in section IV-B, to reduce this complexity to acceptable ranges without sacrificing accuracy.

Tuning the sampling probabilities $p_1, p_2, ..., p_r$, and the number of groups l is closely related to the level of estimation accuracy we would like to achieve. To achieve the *constant relative error tolerance* (discussed in Section II-A), the probabilities are set as $p_i = c^{i-1}$, $i = 1, 2, ..., r$, i.e., a geometric progression. Here $c < 1$ is a constant, which is a function of the number of groups l. The philosophy behind setting parameters this way is captured in Figure 4. Each group covers a certain multiplicity range and in part of this range, it has accurate coverage. When the parameters $p_i's$ are set as above, the accurate coverage ranges of these groups "touch" each other on the borders and jointly cover the whole multiplicity range. In an operating MRSCBF we use throughout the rest of the paper, we set $l = 32$ and $c = \frac{1}{4}$.

This multi-resolution design works very well for Internet traffic, in which the majority of the flows are mice but a small number of large flows (elephants) account for the majority of the packets (the aforementioned "quasi-Zipf" law). Our design ensures that each flow will have a resolution that measures its count with reasonable accuracy. Its storage efficiency is reasonable since the small flows will not occupy too many bits and the bits occupied by large flows will grow only logarithmically with their size. However, MRSCBF pays a little price on storage efficiency for blind streaming, which is that the high multiplicity elements will completely fill up all the high resolution filters so that these filters do not carry much information[1]. Nevertheless, this price is moderate because the fraction of large flows is very small in the Internet traffic.

C. Performance Guarantees

In this section, we evaluate the performance of a MRSCBF configured with aforementioned parameters, according to the three performance metrics discussed in Section II-A, namely, computational complexity, storage complexity, and accuracy. Let k_i be the number[2] of hash functions used in a group belonging to filter i. The computational complexity of the scheme is clearly $\sum_{i=0}^{r} k_i * p_i$ bits per packet. When p_i follows geometric progression as above, this value tends to be small. In our MRSCBF scheme, we set k_1 to 4, and k_2,..., k_r to 6. With other parameters shown above ($l = 32$, $c = \frac{1}{4}$), the total complexity is no more than 6 bits per packet. This would allow us to comfortably support OC-192+ speed using 10ns SRAM. The storage complexity is to a certain extent traffic-dependent. Experimental results show that we achieve high storage efficiency on backbone traffic. For example, on one Tier-1 ISP backbone trace, we found that the storage efficiency is about 2 bits per packet with an SCBF of size 1MB. As to accuracy, our estimates are on the average within 15% of the actual value for flows of all sizes according to mathematics analysis shown in Section IV.

IV. Maximum Likelihood Estimation and Analysis

In this section, we study the mathematics behind the MLE procedure and its accuracy. We also discuss the impact of various design parameters on the complexity-accuracy tradeoff.

[1] Remind that flows with distinct labels hash to different location in the filter array. Though a high multiplicity element fills up the high resolution filter for itself, it does not have any impact at all on the accuracy of the same filter for other elements.

[2] Group sizes can be different from one SCBF to another in MRSCBF.

A. MLE with observations from one SCBF in MRSCBF

We first describe the MLE procedure for one SCBF in a MRSCBF. Let Θ be the set of groups that are matched by an element x in SCBF i. We know from the design of MRSCBF that elements are inserted into SCBF i with sampling probability p_i. To find out the number of occurrences of x from the observation Θ, we use the principle of MLE, i.e., we would like to find f that maximizes $Pr(F = f|\Theta)$. In other words, $\hat{F} = \underset{f}{argmax} Pr(F = f|\Theta)$. However, to compute $Pr(F = f|\Theta)$, we need to prescribe an a priori distribution for F. We found that, when F is assumed to have a uniform a priori distribution, $\underset{f}{argmax} Pr(F = f|\Theta) = \underset{f}{argmax} Pr(\Theta|F = f)$. In this case, MLE using $Pr(F = f|\Theta)$ produces the same value as MLE using $Pr(\Theta|F = f)$! This significantly simplifies the MLE process since $Pr(\Theta|F = f)$ has a closed form solution (albeit sophisticated).

Now we explain why $\underset{f}{argmax} Pr(F = f|\Theta) = \underset{f}{argmax} Pr(\Theta|F = f)$ when F has a uniform a priori distribution. By Bayes' rule, $Pr(F = f|\Theta) = \frac{Pr(\Theta|F=f)*Pr(F=f)}{Pr(\Theta)}$. Since the value $Pr(\Theta)$ on the denominator is a constant, the f that maximizes $Pr(F = f|\Theta)$ has to maximimize $Pr(\Theta|F = f) * Pr(F = f)$, the numerator. When F has uniform *a priori* distribution, $Pr[F = f]$ becomes a constant with respect to f and the result follows.

How to prescribe the default *a priori* distribution (the belief before any observation) has always been a controversial issue in statistics [10]. It is however a widely acceptable practice to use uniform as the default when there are no obviously better choices. Assuming uniform as the default is reasonable also for the following reason. It can be shown quantitatively that the evidence Θ in general significantly outweighs the skew caused by any a priori distribution that is slowly varying. A distribution is slowly varying if $|Pr[F = f] - Pr[F = f + 1]| \leq \epsilon$ when ϵ is a very small constant. Clearly there is no reason to believe that the a priori distribution of F is not slowly varying.

Now that maximizing $Pr(F = f|\Theta)$ becomes maximizing $Pr(\Theta|F = f)$. The following theorem characterizes how to compute $Pr(\Theta|F = f)$. Its proof is involved and omitted due to space limitations.

Theorem 1: Let $\theta = |\Theta|$ and α be the percentage of "1" in the MRSCBF. Then $Pr[\Theta|F = f]$ is equal to

$$\left(1-(1-\alpha)^k\right)^{l-\theta}\sum_{q=0}^{f}\binom{f}{q}p^q(1-p)^{(f-q)}\sum_{\beta=0}^{\theta}\Bigg[\left(\frac{\theta}{l}\right)^q\cdot\binom{\theta}{\beta}\left((1-\alpha)^k\right)^{\beta}\left(1-(1-\alpha)^k\right)^{\theta-\beta}\Bigg\{1-\binom{\theta-\beta}{1}\left(\frac{\theta-1}{\theta}\right)^q+\binom{\theta-\beta}{2}\left(\frac{\theta-2}{\theta}\right)^q-\cdots+(-1)^{\theta-\beta}\binom{\theta-\beta}{\theta-\beta}\left(\frac{\beta}{\theta}\right)^q\Bigg\}\Bigg] \tag{1}$$

B. MLE with observations from multiple SCBF's in MRSCBF

Now we describe the MLE process for MRSCBF. Let Θ_1, Θ_2, ..., Θ_r be the set of groups that are matched by the element x in SCBF 1, 2, ..., r respectively. Since $\Theta_1, \cdots, \Theta_r$ are independent, when independent hash functions are used in SCBF's, we have

$$Pr[\Theta_1, \cdots, \Theta_r|F = f] = \prod_{i=1}^{r} Pr[\Theta_i|F = f] \tag{2}$$

Therefore $MLE(\Theta_1, \cdots, \Theta_r) = \underset{f}{argmax} \prod_{i=1}^{r} Pr[\Theta_i|F = f]$. Note that $Pr[\Theta_i|F = f]$ can be computed from Equation 1.

However, although above MLE decoding formula is in principle correct, it cannot be used in practice since the complexity of precomputing the decoding table is prohibitive. We solve this problem by choosing the observations from the "best three consecutive resolutions", namely, Θ_{j-1}, Θ_j, Θ_{j+1} for a certain j depending on the specific values of $|\Theta_i|$, $i = 1, 2, ..., r$. Our analysis shows that $MLE(\Theta_{j-1}, \Theta_j, \Theta_{j+1})$ will achieve accuracy very close to MLE based on all the observations. The theory behind selecting the "best three" is involved and omitted here due to space limitations.

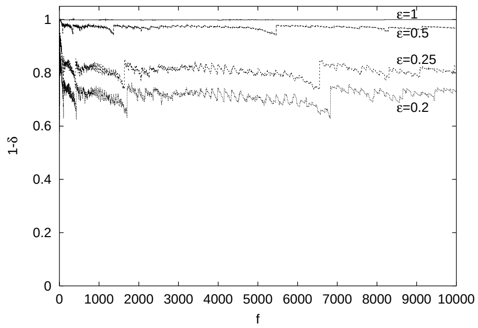

Fig. 5. Probability that the estimate $\hat{F}$ is within a factor of $(1 \pm \epsilon)$ of the actual frequency F for various values of ϵ.

V. Evaluation

A. The accuracy of MLE decoding

The accuracy of estimation by a MRSCBF is a function of the various design parameters, including the number of groups (l_i), the sampling rate (p_i) (resolution), and the number of hash functions (k_i), used in each SCBF i, $i = 1, 2, ..., r$. The accuracy of the MLE decoding for a single group can be characterized by the probability of the estimated value $\hat{F}$ being within the interval $[(1 - \epsilon)F, (1 + \epsilon)F]$, where F is the real value. It can also be characterized as the mean of the difference between the the real and estimated values $E[|\hat{F} - F|]$. Both characterizations can be computed from Equation 2 in a straightforward way.

Figure 5 shows the plot of $(1 - \delta)$ for different values of f, where $1 - \delta = Pr[(1 - \epsilon)F \leq \hat{F} \leq (1 + \epsilon)F]$. The parameters used for the MRSCBF are $r = 9$ virtual SCBFs, $l = 32$ groups in each bloom filter, sampling frequencies of $1, 1/4, 1/16, \cdots, 1/4^{r-1}$ for the r SCBFs and $k = 4$ hash functions per group in the first SCBF and $k = 6$ for the rest. Each curve corresponds to a specific level of relative error tolerance (i.e. a specific choice of ϵ), and represents the probability that the estimated value is within this factor of the actual value. For example, the curve for $\epsilon = 0.25$ shows that about 80% of the time, the estimate is within 25% of the actual value.

The mean of the difference between the the real and estimated values ($E[|\hat{F} - F|]$) is about 15% of the actual value F.

B. Packet header trace measurements

To evaluate the performance of MRSCBF on real-world Internet traffic, we experiment on a set of three packet header traces obtained from a tier-1 ISP backbone. These traces were collected by a Gigascope probe [11] on a high speed link leaving a data center in April, 2003. Among them two were gathered on weekdays and one on a weekend. Each of the packet header traces last few hours and consists of 588$\sim$632 million packet headers and carries 280$\sim$329 GB traffic. The number of unique IP addresses observed in each trace is around 10 million.

We ran MRSCBF on the packet header traces to estimate the length of each flow, identified by either the source or the destination IP address. We observe that MRSCBF is able to process 19$\sim$21 million packets before paging to disk with bloom filter of size 1MB. Figure 6 shows the distribution of actual and estimated flow volume, in terms of the number of packets in the flow, on 20 million packets taken from the packet header traces obtained on April 17, 2003. The MRSCBF over-estimated the total number of packets by about 3%. Similar observations hold on all three packet header traces. Figure 6(a) agrees with our theoretical analysis shown in Figure 5. It shows that MRSCBF achieves a constant relative error tolerance. Each point in the graph corresponds to a flow, with its x coordinate being the actual number of packets in this flow, and its y coordinate the number estimated by MRSCBF. Note that both axes are in logarithm scales. The fact that all the points are concentrated within a narrow band of fixed width along the $y = x$ line indicates that our estimates are consistently within a constant factor of the actual frequency. Figure 6(b) shows the distribution of the original and estimated flow volume (both of them are sorted by the number of packets in a flow). We found that MRSCBF gives near perfect flow volume distribution. This is very useful in applications such as network anomaly detection.

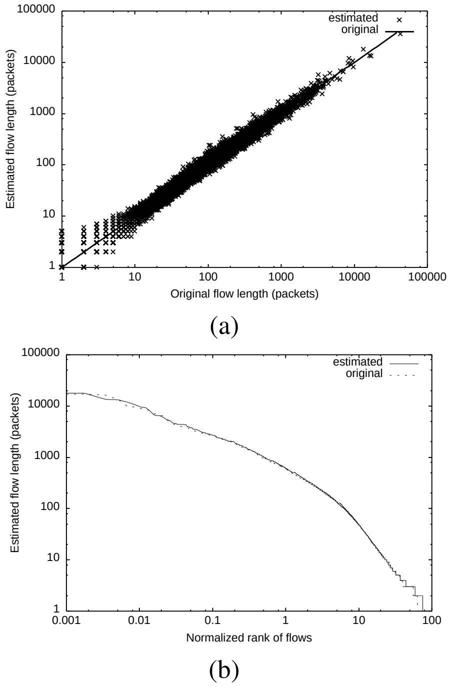

Fig. 6. The original and estimated flow length distributions (both x and y axis are in log scale). (a) Original vs estimated flow length; (b) Distribution of the original and estimated flow length.

VI. Conclusions

Per-flow traffic accounting is important in a number of network applications. However, current solutions such as maintaining per-flow state or random sampling are either not scalable or not accurate. We propose a novel data structure called Space Code Bloom Filter that performs approximate yet reasonably accurate per-flow accounting without maintaining per-flow state. It is very amenable to pipelined hardware implementation since its logic is simple and it is a write-only data structure (blind streaming). We developed procedures for estimating the flow volume from observations of MRSCBF based on the Maximum Likelihood Estimation (MLE) principle. Our analysis shows that our estimation procedure will guarantee constant relative error with high probability. We also experiment MRSCBF and its MLE algorithm on Tier-1 ISP backbone traffic traces. The experimental results agree very well with our theoretical analysis. In our future work, we will explore the fundamental tradeoff between measurement accuracy and complexity, and apply the SCBF data structure to other network measurement problems.

References

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

[2] "http://www.caida.org," .

[3] M. Charikar, K. Chen, and Farach-Colton, "Finding frequent items in data streams," in *ICALP. Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, Germany*, 2002, pp. 693–703.

[4] 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 (TODS)*, vol. 28, pp. 51–55, 2003.

[5] Burton H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *CACM*, vol. 13, no. 7, pp. 422–426, 1970.

[6] L. Fan, P. Cao, J. Almeida, and A.Z. Broder, "Summary cache: a scalable wide-area Web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, 2000.

[7] S. Cohen and Y. Matias, "Spectral bloom filters," in *Proc. ACM SIGMOD Conference on Management of Data*, 2003.

[8] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," in *Proceedings of the ACM Symposium on Theory of Computing*, 1996.

[9] E.D. Demaine, J.I. Munro, and A. Lopez-Ortiz, "Frequency estimation of internet packet streams with limited space," in *European Symposium on Algorithms (ESA). Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, Germany*, 2002.

[10] P. J. Bickel and K. A. Doksum, *Mathematical Statistics, Basic Ideas and Selected Topics*, Prentice Hall, 2001.

[11] Chuck Cranor, Theodore Johnson, and Oliver Spatscheck, "Gigascope: a stream database for network applications," in *Proceedings of SIGMOD 2003*, Jun 2003.