Space-Code Bloom Filter for Efficient Per-Flow Traffic Measurement

Abhishek Kumar*, Jun (Jim) Xu*, Jia Wang†, Oliver Spatschek†, Li (Erran) Li‡

*College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332–0280
{akumar,jx}@cc.gatech.edu

†AT&T Labs - Research
Florham Park, NJ 07932-0971
{jiawang,spatsch}@research.att.com

‡Bell Labs
Lucent Technologies
Holmdel, NJ 07733-3030
erranlli@bell-labs.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". We introduce a novel technique for measuring per-flow traffic approximately, for all flows regardless of their sizes, at very high-speed (say, OC768). 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. The multiplicity of an element in the multiset represented by SCBF can be estimated through either of two mechanisms – Maximum Likelihood Estimation (MLE) or Mean Value Estimation (MVE). 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 evaluate the performance of SCBF through mathematical analysis and through experiments on packet traces gathered from a tier-1 ISP backbone. Our results demonstrate that SCBF achieves reasonable measurement accuracy with very low storage and computational complexity.**

***Index Terms*— Network Measurement, Traffic Analysis, Data Structures, Statistical Inference, Bloom Filter.**

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, network anomaly detection, security, 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) [1], [3], [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 the ratio of false positives, 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 attributes 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.

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-768 (40 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

This work was supported in part by the National Science Foundation under Grant ITR/SY ANI-0113933 and under NSF CAREER Award Grant ANI-0238315.

can easily scale to maintaining approximate per-flow counts at an OC-192 (10 Gbps) or even an OC-768 (40 Gbps) link using a limited amount of fast memory. The storage cost for a fully utilized OC-192 link is tolerable: about 4 bits per packet or 18 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.

Conceptually, an SCBF can be thought of as 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 an estimation procedure. We present two alternative estimation mechanisms, Maximum Likelihood Estimation (MLE) and Mean Value Estimation (MVE). 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* [4], [6]. 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 [7], [6], [8], the techniques invented in the database context tend to be computationally complex, and thus have limited application to problems in networking. 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. The next section presents an overview of the architecture of SCBF and introduces the metrics for measuring its performance. We also introduce the paradigm of blind streaming and discuss its impact on our design. Section III describes the design of SCBF in detail. We provide some essential mathematical details of the design of SCBF in Section IV, deferring the rest to the appendix. Section V presents analytical results on the accuracy of SCBF followed by experimental evaluation of a software implementation using packet header traces from a tier-1 ISP IP backbone network. A brief review of related work is presented in Section VI. We conclude in Section VII.

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, OC-768). A naïve 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 OC-192 since fast SRAM modules can only hold a tiny fraction of per-flow state due to their size limitations, and large DRAM modules cannot support such speed. Random sampling with a small rate such as 1% may meet the speed requirement for keeping the per-flow state in DRAM. However, such sampling leads 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.

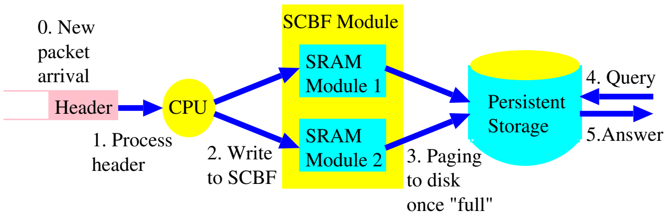


Fig. 1. The system model for using SCBF for traffic measurement.

Our vision is to design a synopsis data structure that keeps track of the approximate 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. Here we describe the conceptual design of SCBF, deferring its detailed description to Section III. 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 alternating 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-768 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 recorded by that SCBF page. The aggregate size of a flow, independent of measurement epochs, can be obtained by taking the sum of flow-size estimates obtained by querying pages corresponding to consecutive epochs.

A. Performance Metrics

The key challenge in 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 OC-768 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 while the majority of flows are small[1].

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 on the average, our scheme will incur no more than 5 bits of write per packet to the memory. We will see later that most of these writes overwrite already written bits, thus filling up the SCBF page at a much slower rate.

3. Accuracy of estimation. We would like our estimation of the traffic-volume of individual flows in a measurement epoch to be as close to the actual value as possible. In this work, our goal is *constant relative error tolerance*, i.e., for the estimate $\hat{F}$ to be within $[(1-\epsilon)F, (1+\epsilon)F]$ with a constant high probability. Here F is the actual value of the traffic-volume of a given flow and $\hat{F}$ is our estimate of this value. We present two estimation mechanisms – Maximum Likelihood Estimation (MLE) and Mean value Estimation (MVE) – to achieve this.

With SCBF, a very high level of accuracy can be achieved if one is willing to incur more complexity in terms of storage and computation. Therefore, there is an inherent tradeoff between the complexities and the accuracy of estimation. This tradeoff can be exploited through the choice of various design parameters, as explained in Section III-C.

B. Blind Streaming

The reader might have noticed that in Figure 1, we do not have an arc from the SCBF module to the CPU. One 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, 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. The loss of accuracy due to this blindness is tolerable, as we will show in Section V.

[1]Our measurement results in Figure 8(b) are an independent verification of this "law".

III. Design Details

A. Space-Code Bloom Filter

At the core of our scheme lies a novel data structure – the Space-Code Bloom Filter (SCBF). It represents a multiset approximately, 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, 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*. During *insertion* , given an element x to be inserted into a set S, the bits $A[h_i(x)]$, $1 \le i \le k$, are set to 1. To *query* for an element y, i.e. to check if 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" even though the set actually 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 probability of encountering a 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 the bits set to 1, and the other half to 0. We refer to this as the "50% golden rule".

In a traditional Bloom filter, once an element x is inserted, later insertions of x will write to the same bits $A[h_1(x)], A[h_2(x)], \cdots, A[h_k(x)]$, 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 $\{h_1^1(x), h_2^1(x), \cdots, h_k^1(x)\}$, $\{h_1^2(x), h_2^2(x), \cdots, h_k^2(x)\}$, $\cdots$, $\{h_1^l(x), h_2^l(x), \cdots, h_k^l(x)\}$. Each group can be viewed as a traditional Bloom filter.

The insertion and query algorithms of SCBF are shown in Figure 2. During insertion, one group of hash functions $\{h_1^i(x), h_2^i(x), \cdots, h_k^i(x)\}$ is chosen randomly, and the bits $A[h_1^i(x)], A[h_2^i(x)], \cdots, A[h_k^i(x)]$ are set to 1. While querying 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 $\{h_1^i, h_2^i, \cdots, h_k^i\}$ if **all** the bits $A[h_1^i(y)], A[h_2^i(y)], \cdots, A[h_k^i(y)]$ are 1. Based on the number of groups that y has matched, denoted as $\hat{\theta}$, we can estimate the multiplicity of y in the multiset, and return the result generated by this estimation procedure as the answer. In other words, a query for the flow y first counts the number of groups matched

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

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

Fig. 2. Insertion and Query in SCBF

Insertion algorithm (given x):
for($j = 1$; $j \leq r$; $j++$)
Insert x into SCBF j with probability p_j
/*shown in Figure 2.*/

Query algorithm (given y):
Check $y's$ occurrences in SCBF 1, 2, ..., r
and obtain counts $\hat{\theta}_1, \hat{\theta}_2, ..., \hat{\theta}_r$ respectively;
return $Estimate(\hat{\theta}_1, \hat{\theta}_2, ..., \hat{\theta}_r)$;

Fig. 3. Insertion and Query Algorithms in MRSCBF

by y and then calls an estimation procedure that uses this count to estimate the multiplicity of y in the multiset. We present two alternative mechanisms for estimating the multiplicity of y from the observation $\hat{\theta}$. The first is Maximum Likelihood Estimation (MLE), which can be thought of as guessing the *most likely* multiplicity of y that would have caused the observation $\hat{\theta}$. The second estimation mechanism is Mean Value Estimation (MVE) which computes that multiplicity of y which is expected to cause the observation $\hat{\theta}$ on the average.

From an implementation perspective, both MLE and MVE can be implemented as a simple lookup into an estimate table precomputed for all possible values of $\hat{\theta}$. Thus each call to $Estimate(.)$ involves a single lookup into a small table. The precomputation of the estimate tables themselves is a computationally intensive procedure but has to be done only once, during the design of the SCBF. We present the theoretical details of MLE and MVE and discuss the precomputation of the estimate tables in Section IV. Although the one-time precomputation of lookup tables is equally involved for both MLE and MVE, we run into precision issues in floating point computation in the case of MLE. Fortunately, these precision problems do not dominate the computation for a range of parameters. Thus we were able to evaluate and compare the levels of accuracy attained by both these mechanisms in Section V.

B. Multi-Resolution Space-Code Bloom Filter

The distribution of flow-sizes in Internet traffic is known to be heavy-tailed, implying that the potential multiplicity of an element can be very high (i.e. some flows are very large). By the *Coupon Collector's problem* [9], all l groups in a SCBF will be used for insertion at least once with high probability after about $(l \ln l)$ copies of x are inserted. In other words almost all elements with multiplicities of $(l \ln l)$ or more will match all the l groups in an SCBF. For example, in an SCBF with 32 groups of hash functions ($l = 32, l \ln l \approx 111$), if 200 copies of x are inserted, it is almost certain that all 32 groups will be chosen at least once. Inserting an additional 200 copies of x will not change anything as all the 32 groups have already been used. Thus any estimation mechanism will be unable to decide whether the number of copies of x inserted in the SCBF was 200 or 400. So, an SCBF with l groups is unable to distinguish between multiplicities that are larger than $(l \ln l)$. 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 actual multiplicity of an element y (signal) is small, the noise will overwhelm the signal. Second, the storage efficiency of the scheme worsens as multiple occurrences of an element are spread to a very large space.

Our solution to this problem is Multi-Resolution SCBF (MRSCBF). An MRSCBF employs multiple SCBFs (or *filters*), operating at different resolutions and uses all of them together to cover the entire range of multiplicities. The insertion and query algorithms for MRSCBF 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 **each** SCBF i with a corresponding sampling probability p_i. Suppose there are a total of r filters (SCBFs). Without loss of generality, we assume $p_1 > p_2 > ... > p_r$. The higher p_i values correspond to higher resolution while lower p_i values imply lower resolution. In this scheme, the 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 other words, filters into which packets are sampled and inserted with high probabilities can keep track of small flows while filters with low sampling probabilities track the larger flows.

In the query algorithm, we count the number of groups that x matches in filters $1, 2, \cdots, r$, denoted as $\hat{\theta}_1, \hat{\theta}_2, \cdots, \hat{\theta}_r$ respectively. Due to the varying sampling probabilities of the individual filters, in general the values $\hat{\theta}_1, \hat{\theta}_2, \cdots, \hat{\theta}_r$ will all be different. The final estimate will be $Estimate(\hat{\theta}_1, \hat{\theta}_2, \cdots, \hat{\theta}_r)$, the result of a joint estimation procedure based on the observations. The first step in this estimation procedure is to identify one or more of the total r filters whose observations are the most relevant for estimation. The theory and mechanism for identifying relevant filters is explained in Section IV-C. Once the relevant filter(s) are identified, a precomputed estimate table can be looked-up using the corresponding observations.

Like in SCBF, the estimate table for MRSCBF again will be precomputed. We developed techniques, discussed in section IV-A.2 and IV-C, to optimally compute the estimate table 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),

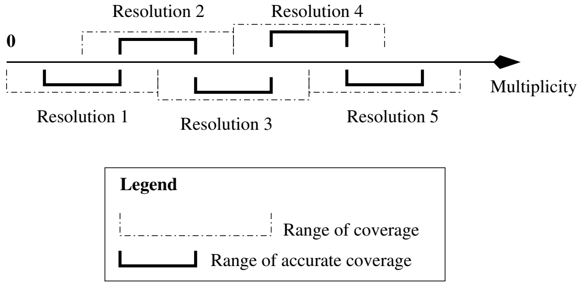


Fig. 4. The conceptual design of MRSCBF

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. With geometrically decreasing sampling probabilities, the absolute inaccuracy in estimation by the corresponding filters also increases geometrically. But filters with small sampling probabilities are only used to estimate the size of large flows, or in other words, have a geometrically larger range of coverage. So the relative accuracy of estimation remains constant. For the analysis and evaluation of MRSCBF throughout the rest of the paper, we set $l = 32$ and $c = \frac{1}{4}$, unless specified otherwise.

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 for each possible flow size, one of the filters will have a resolution that measures its count with reasonable accuracy. The storage efficiency of MRSCBF is reasonable since the small flows are restricted to filters with high sampling probabilities, and hence will not occupy too many bits. Only large flows get sampled into other filters[2], and due to the geometrically decreasing sampling probability, 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[3]. Nevertheless, this price is moderate because the fraction of large flows is very small in the Internet traffic.

[2]Some small flows too might get sampled into filters with small sampling probabilities, but their impact on storage efficiency is negligible. This does not impact the accuracy either, because the mechanism for choosing the most relevant filter(s) identifies such filters as irrelevant.

[3]Recall 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.

C. Performance Guarantees

At this point in our discussion, we are ready to evaluate the performance of MRSCBF according to one of the three metrics discussed in Section II-A – its computational complexity.

Consider an MRSCBF configured with aforementioned parameters ($l = 32$, $c = \frac{1}{4}$). Let k_i be the number of hash functions used in a group[4] belonging to filter i, and let p_i be the sampling probability of filter i. The computational complexity of the scheme is $\sum_{i=0}^{r} k_i * p_i$ bits per packet. When the sampling probabilities p_i follow a geometric progression, as described in section III-B, this value tends to be small. In our experiments with MRSCBF, we set k_1 to 3, k_2 to 4,and $k_3, \cdots, k_r$ to 6. With other parameters shown above , the total complexity is no more than 5 bits per packet. This would allow us to comfortably support OC-768 speed using SRAM with access latency of 5ns. Assuming average packet-size of 1000 bits and full utilization, an OC-768 link would see 40 million packets per second or one packet every 25ns, thus leaving 5ns for each of the 5 bits to be written to bit-addressable SRAM.

The performance of MRSCBF along the other two metrics of storage complexity and accuracy is evaluated in Section V through analysis and experiments on real packet-header traces.

IV. MAXIMUM LIKELIHOOD ESTIMATION AND ANALYSIS

In this section, we formally specify the two estimation mechanisms - Maximum Likelihood Estimation (MLE) (Section IV-A) and Mean Value estimation (MVE) (Section IV-B), and present the mathematics behind the two procedures. We also describe the mechanism used to choose the "most relevant" filter(s) in an MRSCBF (Section IV-C). Finally, we compare the advantages and disadvantages of MLE and MVE in Section IV-D.

A. Maximum Likelihood Estimation

1) MLE with observations from one SCBF: 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).

[4]Group sizes can be different from one SCBF to another in MRSCBF, but we use the same group size across all SCBFs for simplicity.

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 $Pr[\Theta]$ in the denominator is a constant, the f that maximizes $Pr[F = f|\Theta]$ has to maximize $Pr[\Theta|F = f] * Pr[F = f]$, 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 can be found in the Appendix.

Theorem 1: Let $\theta = |\Theta|$, p be the sampling probability and α be the fraction of bits that are set to '1' in the MRSCBF. Then $Pr[\Theta|F = f]$ is equal to

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

2) 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 (Equation 2) is correct in principle, it cannot be used in practice since the complexity of precomputing the decoding table is prohibitive. We return to this issue in Section IV-C, after discussing MVE in the following subsection.

B. Mean Value Estimation

We begin with the description of MVE procedure for a single SCBF. Again, Let f be the multiplicity of an element x. The number of positives (number of matched groups) we observe from the SCBF, when queried for x, is clearly a random variable that is a function of f (not a function of x with good hash functions). We denote this random variable as θ_f. Let $g(f) = E[\theta_f]$ be the function that maps the multiplicity to the average number of positives it will generate. Clearly $g(f_1) < g(f_2)$ when $f_1 < f_2$ since the higher the multiplicity of x, the higher number of positives it will generate. So g^{-1} exists since it is monotonically increasing.

Our MVE estimation rule is the following. Given an observation $\hat{\theta}$, which is the number of groups matched by the flow x that is being queried for, we declare that $MVE(\hat{\theta}) = g^{-1}(\hat{\theta})$. In other words, given the observation $\hat{\theta}$, our estimate is the value of f that, on the average, produces $\hat{\theta}$ positives. We abused the notation g^{-1} here since g^{-1} is only defined on the discrete points by the above definition. However, we can use linear extrapolation to extend both g and g^{-1} to the continuous domain. In the following, we use the notation g, g^{-1} to represent both the strictly defined discrete functions and their continuous counterparts.

The following theorem characterizes the function g.

Theorem 2:

$$g(f) = \sum_{i=0}^{l} \sum_{q=0}^{f} \binom{f}{q} p^q (1-p)^{f-q} \binom{l}{i} (\alpha^k)^i (1-\alpha^k)^{l-i} (i + j(q,i)) \tag{3}$$

where $j(q,i)$ is the value of j that satisfies the equation

$$q \approx \frac{l}{l-i} + \frac{l}{l-(i+1)} + ... + \frac{l}{l-(i+j-1)}$$

.

Proof: Let θ be the random variable that denotes the number of positives obtained on querying the SCBF for x, given that x has been inserted into the SCBF f times. By the definition of g, $g(f) = E[\theta]$. Let R be the random variable that denotes the number of false positives. Let Q be the number of sampled packets that on the average result in j positives that do not overlap with any of the R false positives. We claim that $E[\theta|R = i, Q = q] = i + j(q, i)$, where $q \approx \frac{l}{l-i} + \frac{l}{l-(i+1)} + ... + \frac{l}{l-(i+j-1)}$. To see this, we view this as an instance of the coupon collector's problem, where we want to collect θ unique coupons. Now i different coupons (the false positives) have already been collected, and the question is how many more coupons on the average do we have to collect to bring the total number of unique coupons we collect to θ. It can be verified that the answer to this question is $j(q, i)$, using arguments from the coupon collector's problem [9]. Therefore,

$$E[\theta] = E[E[\theta|R = i, Q = q]]$$
$$= \sum_{i=0}^{l} \sum_{q=0}^{f} Pr[R = i, Q = q] E[\theta|R = i, Q = q]$$
$$= \sum_{i=0}^{l} \sum_{q=0}^{f} \binom{f}{q} p^q (1-p)^{f-q} \binom{l}{i} (\alpha^k)^i (1-\alpha^k)^{l-i} [i + j]$$

■

In an MRSCBF with multiple filters running in parallel, MVE can be used by first identifying the most relevant among the various SCBF (through the mechanism described in the following subsection) and then running MVE for the observation from that SCBF.

C. Choosing the most relevant filters in an MRSCBF

Using multiple SCBFs in an MRSCBF with different, independent sampling probabilities, allows it to account for items with large multiplicities with a constant relative accuracy. But the estimation procedure becomes more complex for both MLE and MVE. Equation 2 shows how the joint probability distribution of multiple filters can be calculated from the probability distribution for individual filters. But the complexity of this operation is l^r where l is the number of possible outcomes in an individual filter and r is the total number of filters. This combinatorial explosion in complexity rules out an implementation of the estimation procedure in this form.

Fortunately, we can solve this problem by using the observations from the "most relevant" filters. The key observation here is that due to the varying sampling probabilities of different independent SCBF's in an MRSCBF, for a given multiplicity, only a few filters have a resolution that is good enough to estimate that particular multiplicity. All other filters have a resolution that is either too fine or too coarse to have any relevance. For example, if an item x has multiplicity of 1000, then the filter sampling with probability 1 (resolution too fine) will have all the groups matched by x (i.e., $\theta = l$), and thus will not be able to provide any information[5] about the multiplicity of x . On the other hand a filter that samples with a probability of $\frac{1}{1024}$ (resolution too coarse) will have $\theta = 0$ or 1 most of the time, when looked up for x. Such a value of $\theta = 0$ or 1 is so small that it will be dominated by the false positives. Intuitively, we can see that filters that sample with probabilities $\frac{1}{16}$, or $\frac{1}{32}$ might provide the best estimates. In our experiments we observed that choosing the filter j^* with the "best resolution" and the two filters $j^* - 1$ and $j^* + 1$, and then using the joint probability distribution for these three filters using (2) was a sound strategy.

We now explain the mechanism used to identify the most relevant filter (i.e., the filter with the most relevant observation for a particular item x). As explained above, filters with very small or very large observations do not provide enough information about the given item. More formally, we can talk about the *relative incremental inaccuracy* of a given observation. For a filter with l groups, θ of which are matched by an item x, it would take about $\frac{l}{l-\theta}$ insertions on the average to match another unmatched group, and increase the observation to $\theta + 1$. On the other hand we know from the coupon collector's problem that the total number of insertions required to cause the observation θ is $(\frac{l}{l} + \frac{l}{l-1} + \cdots + \frac{l}{l-\theta+1})$. Thus the *relative incremental inaccuracy* of this observation is $\frac{l}{(l-\theta)}/(\frac{l}{l} + \frac{l}{l-1} + \cdots + \frac{l}{l-\theta+1})$. To identify the most relevant filter, we calculate the value of *relative incremental inaccuracy* of the observations $\theta_i = |\Theta_i|$ from each of the r windows and choose the filter with the smallest inaccuracy. Figure 5 shows a plot of the relative incremental inaccuracy for different values of θ. Note that the y-axis is on logscale. We can observe in both curves, for filters with 32 and 64 groups respectively, that the inaccuracy is the least when θ is an intermediate value between 0 and the total number of groups. Another observation is that the relative incremental inaccuracy of a filter with 64 groups is better than that of a filter with 32 groups for any observation θ.

[5] All we can determine from an observation $\theta = l$ is that the multiplicity of x is too large to be estimated by this filter.

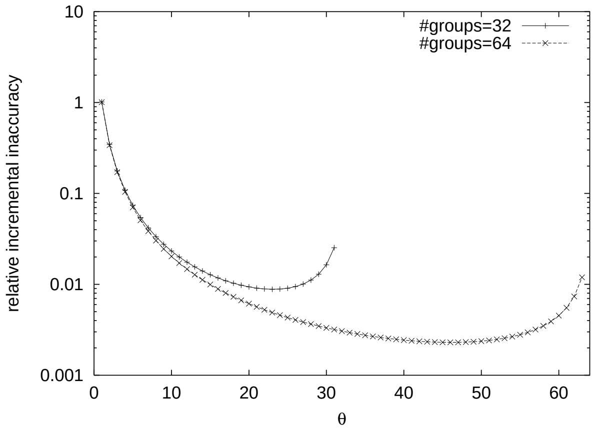
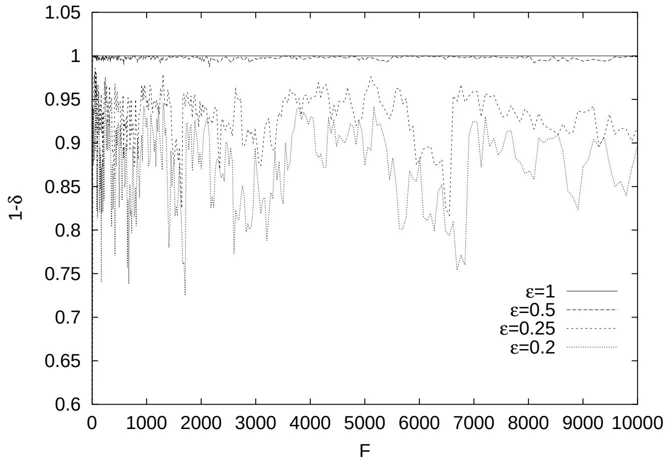


Fig. 5. The relative incremental inaccuracy for filters with 32 and 64 groups. Notice the logscale on the y-axis.

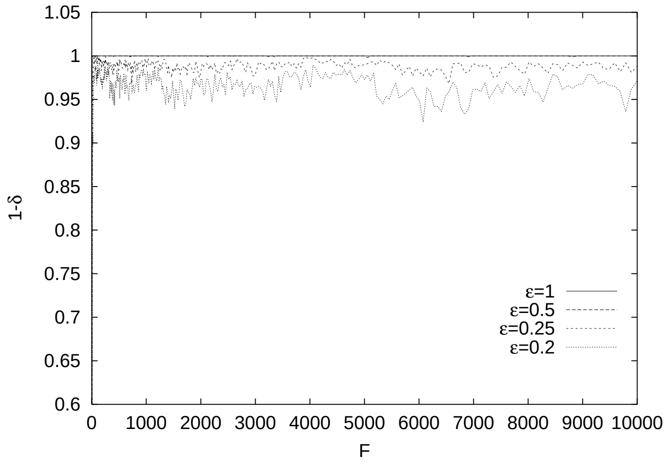
D. Comparison of MLE and MVE

In statistics, MLE is the provably-optimal decoding procedure for minimizing estimation error. However, the error metric in that context is $Pr[F \neq \hat{F}]$, i.e., the probability that the estimate is different from the actual value, given an observation θ. In our application, however, $Pr[F \neq \hat{F}]$ is very close to 1, since many possible $F's$ are almost equally likely to produce the observation θ, and $MLE(\theta)$ is only slightly more likely than many of its neighbors to produce θ ("peak of an almost-flat hill"). So we use $Pr[(\hat{F} - F) \leq \epsilon F]$ as our metric for evaluation. But under this metric, it can be shown that MLE may not give us optimal performance, and MVE can actually perform better, as shown in Section V-A. The decoding table for MVE is also easier to compute than that of MLE, especially when the number of groups is over 35 (due to the floating point precision limit).

However, MLE does have an advantage in our context. Extending MLE for joint estimation from multiple observations is straightforward (Section IV-A.2). On the other hand, it is hard, if not impossible, to find a mathematically sound extention of MVE for multiple observations in our context. So the MLE decoding for MRSCBF is based on three observations, while MVE is only based on the one observation, chosen through the mechanism described in Section IV-C. We will revisit this in Section V-A.



(a) Theoretical accuracy of MVE using 32 groups.



(b) Theoretical accuracy of MVE using 64 groups.

Fig. 6. Probability that the estimate $\hat{F}$ is within a factor of $(1 \pm \epsilon)$ of the actual frequency F for various values of ϵ.

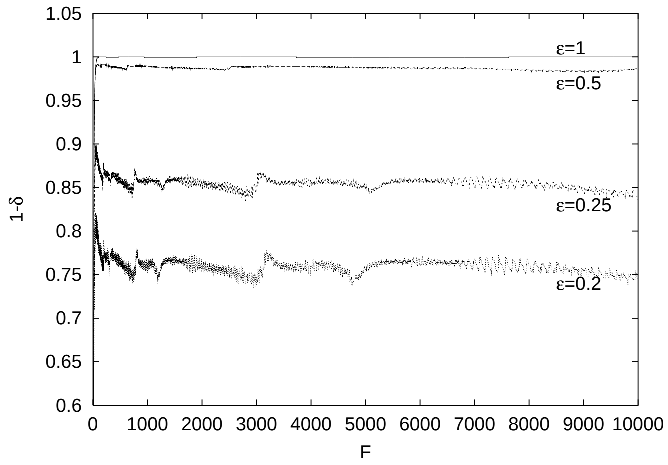


Fig. 7. Probability that the estimate $\hat{F}$ is within a factor of $(1 \pm \epsilon)$ of the actual frequency F for various values of ϵ. MLE using 32 groups.

V. Evaluation

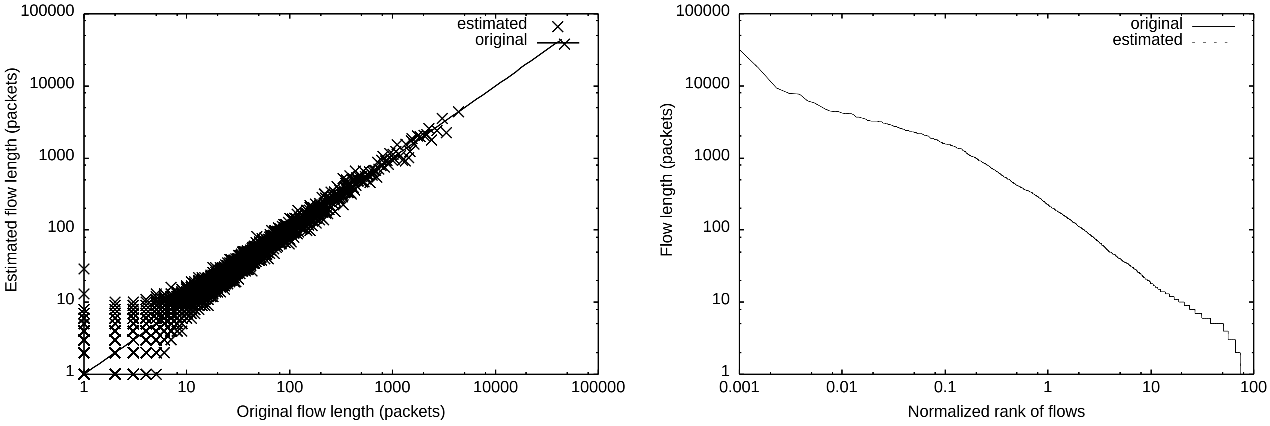
In this section we present an evaluation of MRSCBF using the metrics of accuracy and storage complexity. Section V-A evaluates the *theoretical accuracy* of MRSCBF through analytical calculations only. We then present an evaluation of MRSCBF using real Internet traces in Section V-B. The chief achievement of MRSCBF is accurate estimation of flow sizes as depicted in Figures 8(a) and 8(b) and discussed in Section V-B.2.

A. Theoretical accuracy of Estimation

The accuracy of estimation by MRSCBF is a function of the various design parameters, including the number of filters r, 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 estimation 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 by the mean of relative error ($E\left[\frac{|\hat{F}-F|}{F}\right]$). Both these quantities can be computed from Equation 1.

Figure 7 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 $r = 9$ SCBFs, $l = 32$ groups in each SCBF, sampling probabilities of $1, \frac{1}{4}, \frac{1}{16}, \cdots, \frac{1}{4^{r-1}}$ for the r SCBFs and $k = 3$ hash functions per group in the first SCBF, $k = 4$ for the second 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 around 85% of the time, the estimate is within 25% of the actual value.

Figures 6(a) and 6(b) show similar curves for MVE in filters with 32 and 64 groups respectively. The figures for MVE have been drawn using computations from Theorem 1 (also useful for MVE) and the MVE decoding rule. For the same number of groups (l = 32), we observe that MVE (Figure 6(a)) has very similar accuracy as MLE (Figure 7). However, curves with MLE are much more smooth than those with MVE for the following reason. In MLE, we use the joint observations from three filters for estimation (discussed in Section IV), and this evens out the variations that occur in different windows. However, such a joint estimation procedure does not exist for MVE, and variations in the "most relevant" filter (determined through the mechanism discussed in Section IV-C) reflect directly in Figure 6(a). We have plotted accuracy curves (not shown here due to lack of space) for MLE using a single "most relevant" filter, and the curves are just as "noisy" as the MVE curves in Figure 6(a), thus supporting our interpretation here.



(a) Original vs. estimated flow size. Note that both axes are on logscale.

(b) Distribution of the original and estimated flow size.

Fig. 8. Estimation with MRSCBF using MLE.

We also observe that the accuracy of filters with 32 groups (Figure 6(a)) is lower compared to that of the filter with 64 groups (Figure 6(b)), thus indicating the gain in accuracy with an increase in the number of groups per filter. However this gain comes at the cost of increased storage complexity as the "code" for different flows is spread over a larger "space". We could not compute the accuracy of MLE with filters containing 64 groups of hash functions, because standard floating point libraries will not work for computations involved in Theorem 1 when the window size grows beyond 35 (the computation of MLE decoding table clearly requires the computation of the formula in Theorem 1). We expect to see a similar gain in accuracy with MLE on increasing the number of groups, and plan to verify this using high precision floating point library in the future. In separate calculations for both MLE (32 groups) and MVE (32 and 64 groups), we observed that the mean of relative error ($E\left[\frac{|\hat{F}-F|}{F}\right]$) is less than 0.15, implying that on the average, the error in estimation is no more than 15%.

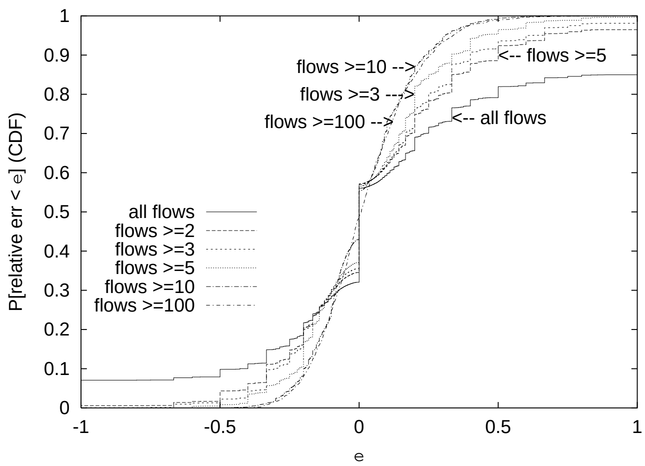


Fig. 9. Accuracy of estimation using MLE. CDF of relative error for flows of various size.

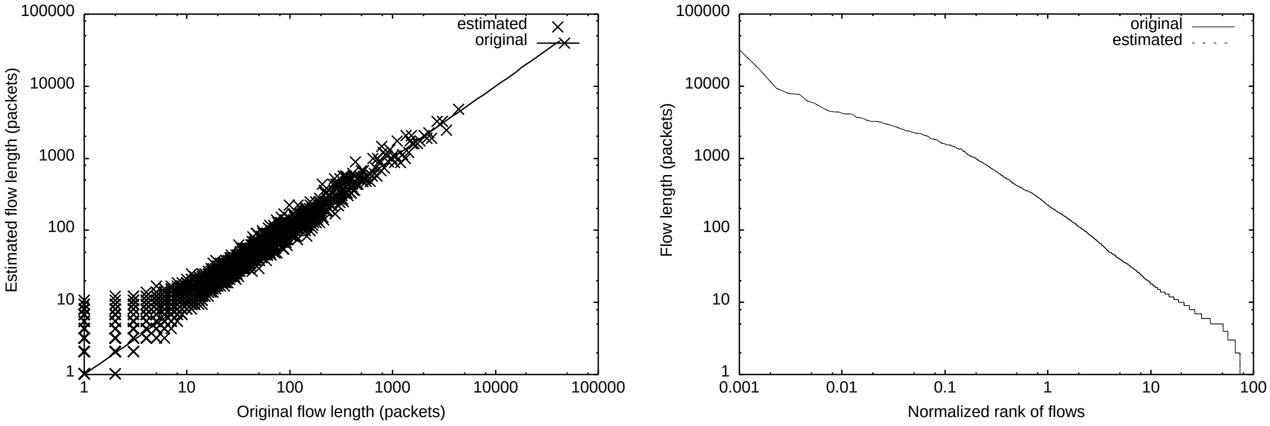
B. Packet header trace measurements

To evaluate the performance of MRSCBF on real-world Internet traffic, we use 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 lasts few hours and consists of 588∼632 million packet headers and carries 280∼329 GB traffic. The number of unique IP addresses observed in each trace is around 10 million.

We developed a software implementation of MRSCBF with both MLE and MVE estimation mechanisms. Throughout the rest of this section, we use the results obtained from processing packet header traces using this implementation. The operation of MRSCBF is independent of the definition of "flow". We evaluate its performance using five popular notions of what constitutes a flow identifier: (i) source IP; (ii) destination IP; (iii) (source IP, destination IP); (iv) (destination IP, destination port); (v) (source IP, source port, destination IP, destination port). We observed similar trends using all five alternatives, and use (source IP, source port, destination IP, destination port) as the flow identifier in the rest of this section to illustrate the measurement results. The figures drawn in this section are all from a trace-segment of about 2 million packets taken from one of the packet header traces obtained on April 17, 2003. Other traces produced virtually identical results.

1) Storage Complexity: With both MLE and MVE, we observed that MRSCBF is able to process[6] over 2 million

[6]The insertion algorithm of MRSCBF is independent of the estimation mechanism.



(a) Original vs. estimated flow size. Note that both axes are on logscale.

(b) Distribution of the original and estimated flow size.

Fig. 10. Estimation with MRSCBF using MVE with 32 groups.

packets before 50% of the bits in a bit-array of size 1MB are set to 1. At this point the array is paged to disk and a new array is initialized and used. If we scale this observation to a fully loaded 10 Gigabit link, with an average packet-size of 1000 bits [12], MRSCBF will need to page 5 MB (uncompressed data) to the disk every second. On the other hand, storing a trace of all packet-headers would take 40 times more space. Note that our estimate of 5 MB of data per second is arrived at with conservative assumptions of very small packets (1000 bits) and 100% link utilization. The average utilization of high capacity links is usually below 50% [13] and the average packetsize is 2 to 6 times higher [14] than our assumption of 1000 bits. Thus we believe that while the worst case storage requirement for MRSCBF is 5 MB per second, the average requirement is quite likely to be an order of magnitude smaller.

2) Maximum Likelihood Estimation: Figure 8(a) shows the scatter diagram of flow size estimated using MLE (y axis) vs. actual flow size (x axis) in terms of the number of packets in the flow. 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 multiplicity. Figure 8(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 MLE gives a near-perfect flow size distribution, indicated by the overlap of the two curves in this figure. This is very useful in applications such as network anomaly detection.

Figure 9 shows the cumulative distribution of relative error. Different curves in this figure correspond to flows that are larger than a particular size. We observe that about 30% of the time MLE estimates the correct flow-size exactly. We also observe that the relative accuracy of MLE is significantly better if we consider only the flows with actual size 10 or more. The reasons for this are twofold. First, we use only 3 hash functions in each group in the first SCBF (with sampling probability 1) to improve the overall computational complexity, and thus incur more false positives in the first filter, leading to poor accuracy for small flows. Second, MLE only returns integer estimates which magnifies the relative error at small absolute values. MLE is not unbiased and tends to slightly over-estimate flow size. During estimation using MLE, the MRSCBF over-estimated the total number of packets by about 3%. Similar observations hold on all three packet header traces.

3) Mean Value Estimation: We ran MRSCBF with MVE on the same sets of packet header traces. We evaluated filters with 32 and 64 groups keeping the remaining parameters same as before. We observed that like MLE, MVE also achieves a constant relative error tolerance and produces a near-perfect distribution of flow sizes. Figures 10(a) and 10(b) illustrate these results. It can be seen that these figures are virtually identical to Figures 8(a) and 8(b) presenting similar results for MLE. However, unlike MLE, MVE is close to unbiased, and over-estimated the total number of packets by only about 0.3%. This is 10 times less than what we observed from MLE. In our experiments, we observe that increasing the number of groups in MVE will improve the accuracy of estimation for larger flows, which again matches our expectations from Section V-A and is consistent with our results on relative incremental inaccuracy in Section IV-C.

Figure 11 shows the cumulative distribution of the relative error of flow size distribution using a filter of 32 groups. The slight bias in the curves for MLE in Figure 9 compared to the roughly symmetric nature of the curves in Figure 11 is another manifestation of the bias of MLE towards over-estimation.

VI. Related Work

Burton Bloom [5] designed the Bloom filter as a space-efficient data structure for answering set-membership queries with a certain probability of false-positives that can be made

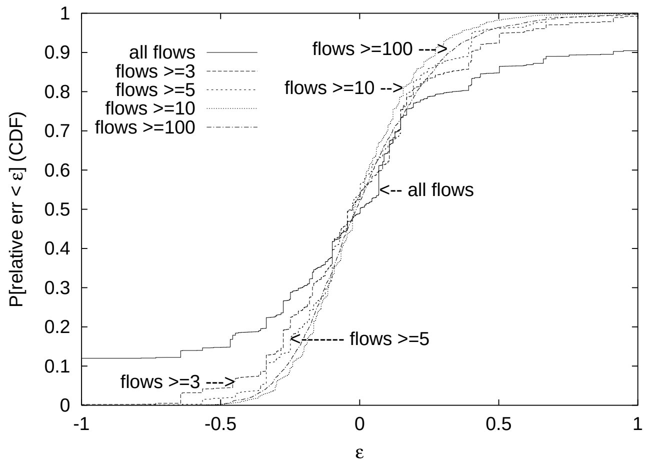


Fig. 11. The cumulative distribution of relative error in flow size estimation using MVE with 32 groups.

arbitrarily small. Bloom filters have found application in numerous areas of Computer Science, most notably in database applications, and more recently, in networking. A thorough survey of *Network Applications of Bloom Filters* is presented in [15].

Previous attempts at using Bloom filters to answer multiset queries have produced a number of variations of *counting Bloom filter* [16]. 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, due to collisions in hashing, 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 unlikely to match the rate of a high speed link.

Estan and Varghese present algorithms in [1] that can identify and monitor a small number of elephants with a small amount of fast memory. The approach in [1] is to use a fast mechanism to identify packets belonging to large flows and then maintain per-flow state for such packets only. While this work successfully addresses the problem of tracking the largest few flows, monitoring just a few large flows precludes a range of applications that would be served better with approximate monitoring of all flows.

VII. 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 develop two procedures for estimating the flow volume from observations of SCBF based on the principles of Maximum Likelihood Estimation (MLE) and Mean Value Estimation (MVE) respectively. Our analysis shows that our estimation procedure will guarantee constant relative error with high probability. Experiments with a software implementation of SCBF and both estimation algorithms on traffic traces from a Tier-1 ISP backbone agree very well with our theoretical analysis.

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

[6] 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.

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

[8] E. Demaine, J. 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.

[9] R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge University Press, 1995, pp. 57–63.

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

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

[12] C. Partridge, P. P. Carvey, E. Burgess, I. Castineyra, T. Clarke, L. Graham, M. Hathaway, P. Herman, A. King, S. Kohalmi, T. Ma, J. Mcallen, T. Mendez, W. C. Milliken, R. Pettyjohn, J. Rokosz, J. Seeger, M. Sollins, S. Storch, B. Tober, and G. D. Troxel, "A 50-gb/s ip router," *IEEE/ACM Transactions on Networking (TON)*, vol. 6, no. 3, pp. 237–248, 1998.

[13] S. Iyer, S. Bhattacharyya, N. Taft, and C. Diot, "An approach to alleviate link overload as observed on an IP backbone," in *Proceedings of IEEE Infocom, San Francisco*, Mar. 2003.

[14] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot, "Packet-level traffic measurement from the sprint IP backbone," *IEEE Network Magazine*, Nov. 2003.

[15] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," in *Fortieth Annual Allerton Conference on Communication, Control, and Computing*, 2002.

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

APPENDIX

We first describe how a single Bloom filter with multiple groups of hash functions (i.e. a SCBF) can be queried to obtain an estimate of the number of insertions in the Bloom filter. We then extend this result to address the possibility of false positives and the impact of sampling before insertion. Finally, we combines these results to complete the proof of Theorem 1.

A. MLE of a SCBF

Le Q be the random variable representing the number of elements inserted in the SCBF. All the events defined henceforth are conditioned on the event $Q = q$. Define[7] A as the event that a particular set of groups of hash functions $\Theta = \{G_1, G_2, \cdots G_\theta\}$ are matched. Let $\theta = |\Theta|$. Event A happens if and only if events B and C happen, where event B is the event that each of the q elements chooses one group from Θ when the element is inserted into the SCBF, and C is the event that all groups of hash function in Θ are chosen. Then $Pr[A] = Pr[B \cap C] = Pr[C|B]Pr[B]$ where $Pr(B) = (\frac{\theta}{l})^q$.

Define A_{i_1} as the event that for each of the q insertions, the group G_{i_1} was never chosen given that all the q insertions chose one of the θ groups in Θ. We claim that $P[A_{i_1}] = \left(\frac{\theta-1}{\theta}\right)^q$ for any i_1, $P[A_{i_1} \cap A_{i_2}] = \left(\frac{\theta-2}{\theta}\right)^q$ for any $i_1 < i_2$ and so on. Now, the probability $Pr[C|B]$ is the same as the probability that none of the events A_i happens, i.e. $Pr[C|B] = Pr[\overline{\bigcup_{i=1}^{j} A_i}]$. Therefore, we have

$$Pr[A] = \left(\frac{\theta}{l}\right)^q Pr\left[\overline{\bigcup_{i=1}^{j} A_i}\right] \tag{4}$$

By the principle of inclusion and exclusion, we obtain the following lemma.

Lemma 1: The probability that the set Θ groups of hash functions is chosen given q insertions of an item is given by: $Pr[A] = \left(\frac{\theta}{l}\right)^q \left[1 - \binom{\theta}{1}\left(\frac{\theta-1}{\theta}\right)^q + \binom{\theta}{2}\left(\frac{\theta-2}{\theta}\right)^q + \cdots + (-1)^{\theta-1}\binom{\theta}{\theta-1}\left(\frac{1}{\theta}\right)^q\right]$

B. Impact of false positives

We have assumed that false positives are negligible in the proof of Lemma 1. In the following we incorporate the effect of false positives into our equation. Suppose the fraction of "1" in the array that holds all the SCBFs is α. Then the probability of false positive in one of the virtual Bloom filters with k_i hash functions in each group is α^{k_i}. The proof of lemma 1 can now be suitably modified to reflect the possibility of false positives.

Lemma 2 (Decoding with false positives): Let B be a Space-code Bloom filter such that elements of a multiset of size n are inserted using one of l groups of k hash functions. Let α be the fraction of '1's in the SCBF. The probability that the set Θ groups of hash functions returns positive given q insertions of an item is given by:

$$Pr[A] = (1-\alpha^k)^{l-\theta}\left(\frac{\theta}{l}\right)^q \sum_{\beta=0}^{\theta}\Bigg[\binom{\theta}{\beta}\left(\alpha^k\right)^\beta (1-\alpha^k)^{\theta-\beta} \left\{1 - \binom{\theta-\beta}{1}\left(\frac{\theta-1}{\theta}\right)^q + \binom{\theta-\beta}{2}\left(\frac{\theta-2}{\theta}\right)^q - \ldots + (-1)^{\theta-\beta}\binom{\theta-\beta}{\theta-\beta}\left(\frac{\beta}{\theta}\right)^q\right\}\Bigg] \tag{5}$$

Proof: By elementary probability theory, we get

$$Pr[A] = \sum_{\beta=0}^{\theta} Pr[A|R=\beta]Pr[R=\beta] \tag{6}$$

where, $R = \beta$ represents the event that there are β false positives out of θ virtual Bloom filters. From the above discussion on false positives, we know that the probability of a false positive is α^k. Using this, we get the probability of β false positives out of θ positives:

$$Pr[R=\beta] = \binom{\theta}{\beta}\left(\alpha^k\right)^\beta (1-\alpha^k)^{\theta-\beta} \tag{7}$$

Suppose the set of virtual Bloom filters that cause false positives are associated with the groups of hash functions G_{f_1}, G_{f_2}, ..., G_{f_β}. Suppose the remaining $\theta - \beta$ groups of hash functions are $G_{t_1}, G_{t_2}, \cdots, G_{t_{\theta-\beta}}$. $Pr[A|R=\beta]$ is the probability that the q packets have been inserted using **at least** all the remaining $\theta - \beta$ groups. Note that "false positive" is a misnomer here — it only means that each of these filters returns positive (when queried) even without insertions of these q packets. So it is possible that some of the q packets chose one or more of the G_{f_i} when they are inserted.

We conditioned all probability on the fact that all q packets used one of the θ groups. Let B_i be the event that the groups of hash functions G_{t_i}, $i = 1, 2, \cdots, \theta-\beta$ are never used. It is easy to see that, $Pr[B_i] = (\frac{\theta-1}{\theta})^q$, $Pr[B_i \bigcap B_j] = (\frac{\theta-2}{\theta})^q$, $\cdots$,$Pr[B_1 \bigcap B_2 \bigcap \cdots \bigcap B_{\theta-\beta}] = (\frac{\beta}{\theta})^q$.

$Pr[A|R = \beta]$ is the probability that, there are no positives other than the θ positives and each of the q insertions chooses one of the θ groups of hash functions in Θ. Therefore, We have $Pr[A|R = \beta] = (1-\alpha^k)^{(l-\theta)}(\frac{\theta}{l})^q Pr\left[\overline{B_1 \bigcup B_2 \bigcup \cdots \bigcup B_{\theta-\beta}}\right]$. By the principle of inclusion and exclusion, we get

$$Pr[A|R=\beta] = (1-\alpha^k)^{l-\theta}\left(\frac{\theta}{l}\right)^q \cdot \left\{1 - \binom{\theta-\beta}{1}\left(\frac{\theta-1}{\theta}\right)^q + \binom{\theta-\beta}{2}\left(\frac{\theta-2}{\theta}\right)^q - \ldots + (-1)^{\theta-\beta}\binom{\theta-\beta}{\theta-\beta}\left(\frac{\beta}{\theta}\right)^q\right\} \tag{8}$$

Substituting the terms in equation 6 using equations 7 and 8 completes the proof. ∎

After incorporating the effect of sampling in the decoding mechanism we obtain the following equation:

$$Pr[\Theta|F=f] = \sum_{q=0}^{f} Pr[A(q)]\binom{f}{q}p^q(1-p)^{(f-q)} \tag{9}$$

See footnote 7 for clarification on $A(q)$. Substituting equation 5 into equation 9 completes the proof of Theorem 1.

[7]Strictly speaking, we should call this event $A(q)$ because it is conditioned upon the event $Q = q$. In the interest of simplicity, we use A in the place of $A(q)$ everywhere except equation 9 where this conditioning needs to be made explicit.