Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Size Distribution

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

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

ABSTRACT

Knowing the distribution of the sizes of traffic flows passing through a network link helps a network operator to characterize network resource usage, infer traffic demands, detect traffic anomalies, and accommodate new traffic demands through better traffic engineering. Previous work on estimating the flow size distribution has been focused on making inferences from sampled network traffic. Its accuracy is limited by the (typically) low sampling rate required to make the sampling operation affordable. In this paper we present a novel data streaming algorithm to provide much more accurate estimates of flow distribution, using a "lossy data structure" which consists of an array of counters fitted well into SRAM. For each incoming packet, our algorithm only needs to increment one underlying counter, making the algorithm fast enough even for 40 Gbps (OC-768) links. The data structure is lossy in the sense that sizes of multiple flows may collide into the same counter. Our algorithm uses Bayesian statistical methods such as Expectation Maximization to infer the most likely flow size distribution that results in the observed counter values after collision. Evaluations of this algorithm on large Internet traces obtained from several sources (including a tier-1 ISP) demonstrate that it has very high measurement accuracy (within 2%). Our algorithm not only dramatically improves the accuracy of flow distribution measurement, but also contributes to the field of data streaming by formalizing an existing methodology and applying it to the context of estimating the flow-distribution.

Categories and Subject Descriptors

C.2.3 [COMPUTER-COMMUNICATION NETWORKS]: Network Operations - Network Monitoring
E.1 [DATA STRUCTURES]

General Terms

Algorithms, Measurement, Theory

SIGMETRICS/Performance'04, June 12–16, 2004, New York, NY, USA.

Keywords

Network Measurement, Traffic Analysis, Data Streaming, Statistical Inference

1. INTRODUCTION

The problem of estimating flow distribution on a high-speed link has received considerable attention recently [1, 2, 3, 4, 5, 6]. In this problem, given an arbitrary flow size s, we are interested in knowing the number of flows that contain s packets, within a monitoring interval. In other words, we would like to know how the total traffic volume splits into flows of different sizes. An estimate of the flow distribution contains knowledge about the number of flows for all possible flow sizes, including elephants (large flows), "kangaroos/rabbits" (medium flows), and "mice" (small flows).

1.1 Motivation

Flow distribution information can be useful in a number of applications in network measurement and monitoring[1]. First, flow distribution information may allow service providers to infer the usage pattern of their networks, such as the approximate number of users with dial-up or broadband access. Such information on usage patterns can be important for the purpose of pricing, billing, infrastructure engineering, and resource planning. In addition, network operators may also infer the type of applications that are running over a network link without looking into the details of traffic such as how many users are using streamed music, streamed video, and voice over IP. In the future, we expect more network applications to be recognizable through flow distribution information.

Second, flow distribution information can help locally detect the existence of an event that causes the transition of the global network dynamics from one mode to another. An example of such mode transition is a sudden increase in the number of large flows (i.e., elephants) in a link. Possible events that may cause this include link failure or route flapping. Merely looking at the total load of the link may not detect such a transition since this link could be consistently heavily used anyway.

Furthermore, flow distribution information may also help us detect various types of Internet security attacks such as DDoS and Internet worms. In the case of DDoS attacks, if the attackers are using spoofed IP addresses, we will observe a significant increase in flows of size 1. In the case of Internet

[1]Here we minimized the overlap with the motivation provided in [1].

worms, we may suddenly find a large number of flows of a particular size in Internet links around the same time, if the worm is a naive one that does not change in size. Also, the historical flow distribution information stored at various links may help us study its evolution over time.

Finally, knowing the flow distribution of each link may help other network measurement applications such as traffic matrix estimation [7, 8, 9, 10]. Recent work [9, 10] show that it is possible to use tomography techniques to infer the traffic matrix from link load and aggregate input/output traffic at each node. We have preliminary evidence to believe that flow distribution at each node will make such tomography much more accurate, since it allows the correlation of not only the total traffic volume (load), but also the correlation of its distribution into different flows.

1.2 Problem statement

The problem of computing the distribution of the sizes of the flows can be formalized as follows. The set of possible flow sizes is the set of all positive integers between 1 to z. Here z is the maximum flow size that can be determined from the observed data. We denote the total number of flows as n, and the number of flows that have i packets as n_i. We denote the fraction of flows that have i packets as ϕ_i, i.e., $\phi_i = \frac{n_i}{n}$. The data that need to be estimated are the values of n and $\phi = \{\phi_1, \phi_2, \cdots, \phi_z\}$. Our goal is to find an efficient scheme to estimate this flow distribution information on a high-speed link (e.g., OC-192 to OC-768) with high accuracy.

A naive solution to this problem is to use a hash table of per-flow counters to keep track of all active flows. These counters will later be examined to obtain the flow distribution. Although this approach is straightforward, it is not suitable for a high-speed link for the following reasons. Each flow entry in the hash table is large (∼160 bits) because it needs to store flow labels (∼100 bits), a pointer (∼ 32 bits) to the next entry if chaining is used to resolve hash collision[2], and a packet counter (∼32 bits). Since there can be a large number of flows (e.g., 0.5 million) on backbone links during a typical measurement period, a hash table of this size typically can only fit into DRAM. However, DRAM speed cannot keep up with the link rate of OC-192 and higher[3].

Another possible approach [1] is to sample a small percentage of packets and then infer the flow distribution from the sampled traffic. The algorithm proposed in [1] may well be the best algorithm in getting as much information from the sampled data as possible. However, its accuracy is limited by the typically low sampling rate (e.g., 1%) required to make the sampling operation affordable. Recent work [2] has provided theoretical insights into the limitation of inferring flow distribution from sampled traffic.

1.3 Our approach and contributions

The main contribution of this paper is a novel data streaming algorithm to provide much more accurate estimates of flow distribution. Our algorithm uses a "lossy data structure" that consists of an array of counters. Its total size is small enough to fit easily in fast SRAM. For each incoming packet, our algorithm only needs to increment one underlying counter (in SRAM), making the algorithm fast enough even for 40 Gbps (OC-768) links. The data structure is lossy in the sense that, due to collision in hashing, sizes of multiple flows may be accumulated in the same counter. Therefore, the raw information obtained from the counters can be far away from the actual flow distribution. Our algorithm then uses Bayesian statistical methods such as Expectation Maximization (EM) to infer the most likely flow size distribution that results in the observed counter values after collision. Experiments of this algorithm on a number of large traces demonstrate that it has very high measurement accuracy (within 2% relative error).

However, to achieve this level of accuracy, our algorithm needs to know the approximate (± 50%) value of n, the total number of flows, in order to provision sufficient number of counters for streaming. Provisioning for the worst case (i.e., when the number of concurrent flows are the largest) leads to unnecessary waste of precious SRAM resource in the average case. To address this challenge, we propose a multi-resolution variant of our algorithm that uses a small and fixed amount of SRAM and does not require any prior knowledge about the approximate range of n. It guarantees high accuracy in the average case and graceful degradation in accuracy in the worst case.

Our algorithm not only dramatically improves the accuracy of flow distribution measurement, but also contributes to the field of data streaming by formalizing an existing yet implicit methodology, and exploring it in a new direction. Data streaming [11] 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. In designing this algorithm, we formalize the following methodology.

Lossy data structure + Bayesian statistics = Accurate streaming

Its main idea is to first perform data streaming at very high speed in a small memory to get the streaming results that are lossy. There are two causes for this loss to be inevitable. First, due to the stringent computational complexity requirement of the application (e.g., $25ns$ per packet when processing OC-768 traffic), the streaming algorithm does not have enough processing time to "put the data into the exact place". Second, the streaming algorithm does not have enough space to store all the relevant data. Due to the loss, the streaming result is typically far away from the information we would like to estimate. Bayesian statistics is therefore used to recover information from the streaming result as much as possible. While Bayesian statistics is typically used in existing streaming algorithms to recover the second cause of loss, our algorithm uses it mainly to recover the first cause of loss. Also, to the best of our knowledge, our algorithm is the first to use sophisticated Bayesian tools such as EM in this recovery.

The rest of this paper is organized as follows. In the next section, we provide an overview of the data collection portion of our solution and describe the design of our streaming

[2]Linear probing and double hashing will not help save space since there is a tradeoff between the occupancy ratio and probe length.

[3]With an average packet size of 1000 bits, per-packet processing time can be no more than 100 ns and 25 ns, for OC-192 and OC-768, respectively. A hash table operation in DRAM will take hundreds of nanoseconds due to the need to retrieve the correct flow entry, compare the flow labels, and increment and write back the counter.

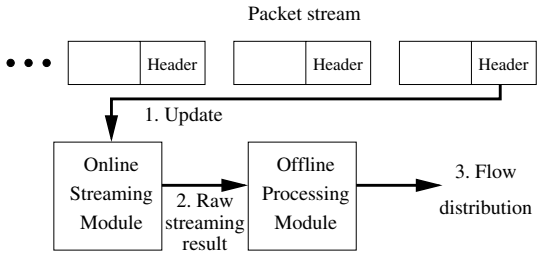


Figure 1: System model of using data-streaming to estimate flow distribution.

1. **Initialize**
2. $A[i] := 0,\ i = 1, 2, ..., m$

3. **Update**
4. Upon the arrival of a packet pkt
5. $ind := \text{hash}(pkt.flow\_label)$;
6. $A[ind] := A[ind] + 1$;

7. **Export data when an epoch ends**
8. $y_j :=$ number of $j's$ in A, $j = 1, 2, ..., z$;
9. Forward the value $y_i's$ to offline analysis;

Figure 2: Algorithm for updating the online streaming module

data structure in detail. Section 3 describes our estimation mechanism. We formalize the estimation mechanism and analyze its correctness in Section 4. Section 5 presents a multi-resolution version of our mechanism that can operate with an array of fixed size. Section 6 evaluates the proposed scheme over a number of large packet header traces obtained from various places including a tier-1 ISP backbone network. We present a brief look at related work with a discussion about the context of our work in Section 7 before concluding in Section 8.

2. DATA STREAMING USING A LOSSY DATA STRUCTURE

In this section, we first give an overview of the system model and the design philosophy of our approach. Then we describe our online update scheme (i.e., the "lossy data structure") and analyze its computational and storage complexity. Finally, we show how our scheme interfaces with the technique in [12] to reduce the storage complexity.

2.1 System model

The overall architecture of our solution is shown in Figure 1. The online streaming module is updated upon each packet arrival (arc 1 in Figure 1). The measurement proceeds in epochs. At the end of each measurement epoch, the counter values, which we refer to as the *raw data*, will be paged out from the online streaming module, and these counters will be reset to 0 for the next measurement epoch. This raw data will be processed by an offline processing module (arc 2 in Figure 1) that produces a final estimate (arc 3 in Figure 1) of the flow distribution[4] using statistical inference techniques. This system model reflects our aforementioned design philosophy of collecting as much pertinent information as possible at the streaming module, and then compensating for the information loss during data collection using Bayesian statistics.

2.2 Online streaming module

Our algorithm for updating the data-streaming module upon packet arrivals is shown in Figure 2. The streaming data structure used by our mechanism is extremely simple – an array of counters. Upon arrival of a packet at the router, its flow label[5] is hashed to generate an index into this array, and the counter at this index is incremented by 1. Collisions due to hashing might cause two or more flow labels to be hashed to same indices. Counters at such an index would contain the total number of packets belonging to all of the flows colliding into this index. We do not have any explicit mechanisms to handle collisions as any such mechanism would impose additional processing and storage overheads that are unsustainable at high speeds. This makes the encoding process very simple and fast. Efficient implementations of hash functions [13] allow the online streaming module to operate at speeds as high as OC-768 without missing any packets.

2.3 Complexity of online streaming module

In this section, we discuss the storage and computational complexities of operating the data streaming module.

1. Storage complexity. This refers to both the amount of fast memory required for implementing the array of counters, and the amount of space (in DRAM or disk) to store the raw counter values for later retrieval and estimation by the offline estimation module. Leveraging on the techniques for efficient implementation of a counter array proposed in [12], we require **9 bits of SRAM per counter** (to be discussed in Section 2.4). This allows us to implement about 1 million counters with 1.1 MB of SRAM.

Interestingly, this raw data can be summarized to a very small size when paged to DRAM or disk. The key fact here is that our estimation mechanism does not need to know the mapping between counter values and indices. Instead, it only needs to know, for each possible counter value, the number (i.e., frequency) of counters that have this value. Therefore, we can summarize this raw data into a list of <*counter value, frequency*> tuples. It turns out that, while the number of flows is large, the unique flow sizes (and consequently, unique counter values) are usually quite small. For example, in a trace with 2.6 million packets and 192,000 flows, we observed only about 500 unique counter values. This implies that most counter values do not occur (i.e., occur with a frequency of zero) in the array, resulting in a very small list of <*counter value, frequency*> tuples. For the above example, the summary can be stored in 8KB on persistent storage, thus requiring less than 0.025 bits per packet, or 1 bit for 40 packets.

2. Computational complexity. For each packet, the data streaming module needs to compute exactly one hash

[4]In practice, the raw data collected at the streaming module can also be summarized and paged to persistent storage, where it can be stored till subsequent retrieval and estimation.

[5]Our design does not place any constraints on the definition of flow label. It can be any combination of fields from the packet header.

function and increment exactly one counter. This is manageable even at OC768 (40 Gbps) speeds with off-the-shelf $10ns$ SRAM. We will show that our efficient (compact) implementation of counters (discussed in Section 2.4) causes very little overhead, allowing operation at OC-768 speed.

2.4 Efficient implementation of an array of counters

Internet traffic is known to have the property that a few flows can be very large, while most other flows are small. Thus, the counters in our array need to be large enough to accommodate the largest flow size. On the other hand, the counter size needs to be made as small as possible to save precious SRAM. Recent work on efficient implementation of statistical counters [12] provides an ideal mechanism to balance these two conflicting requirements, which we will leverage on in our scheme. For each counter in the array, say 32 bits wide, this mechanism uses 32 bits of slow memory (DRAM) to store a large counter and maintains a smaller counter, say 7 bits wide, in fast memory (SRAM). As the counters in SRAM exceed a certain threshold value (say 64) due to increments, it increments the value of the corresponding counter in DRAM by 64 and resets the counter in SRAM to 0. There is a 2-bit per counter overhead that covers the cost of keeping track of counters above the threshold, bringing the total number of bits per counter in SRAM to 9. For suitable choices of parameters, this scheme allows an efficient implementation of wide counters using a small amount of SRAM. This technique can be applied seamlessly to implementing the array of counters required in our data streaming module. In our algorithm[6], the size of each counter in SRAM is 9 bits and in DRAM is 32. Also, since the scheme in [12] incurs very little extra computational and memory access overhead, our streaming algorithm running on top of it can still achieve high speeds such as OC-768.

3. ESTIMATION MECHANISMS

In this section, we describe a collection of estimation mechanisms used in the offline processing module (shown in Figure 1). They help to infer the actual flow distribution from the counter values collected by the online streaming module. Consider the hypothetical case where there are no hash collisions. In this case the distribution of counter values is the same as the actual flow distribution. However, collisions do occur with real-world hash functions, thus distorting the distribution of counter values away from the true flow distribution. This effect[7] is shown in Figure 3, where we process a traffic trace (with 560K flows in it) on arrays of 1024K, 512K, 256K, and 128K counters, respectively. We can see that, as the "load factor" (formally defined later in this section) of the array increases, the number of collisions increases, which further exacerbates this distortion.

3.1 Estimating the total number of flows

The first quantity that we can estimate from our counter array is the total number of flows during the measurement interval. The first mechanism for estimating this quantity (in a different application) using a (0-1) bitmap is proposed in [14]. It can be used in our context with slight adaptation. The process of inserting (with collisions) flow counts into our counter array can be modeled as a coupon collector's problem, under the assumption of uniform hashing. As shown in [14], in an array of m counters, if the number of zero entries is m_0 after the insertion of n flows, the maximum likelihood estimator for n is

$$\hat{n} = m \ln \left(\frac{m}{m_0} \right) \tag{1}$$

This result is also exploited in [6] to design a more general multi-resolution bitmap scheme to estimate n using much smaller memory.

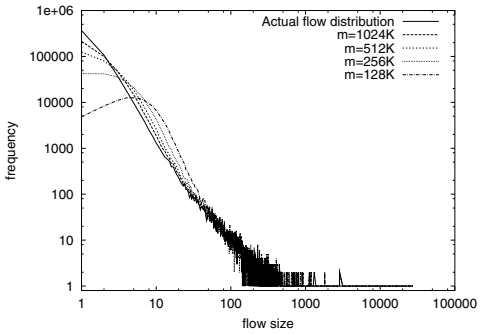


Figure 3: The distribution of flow sizes and raw counter values using varies number of counters (both x and y axes are in log-scale). m = number of counters.

3.2 Estimating the number of flows of size 1

The total number of flows containing exactly one packet is arguably the most significant single piece of information hidden in the distribution of flow sizes. From a modeling perspective, this number helps affirm or reject statistical hypotheses such as whether the flow size distribution is Zipfian. More importantly, abnormal or malicious behavior in the Internet, such as port-scanning and DDoS attacks, often manifests itself as a significant increase in the number of flows of size 1.

To estimate the number of flows of size 1 (denoted by n_1), let us look at the process of inserting flow counts into the counter array. Note that a counter of value 1 must contain exactly one flow of size 1 (i.e., no collision). Based on this insight, we can derive a very accurate estimator for n_1. Let $\hat{\lambda} = \frac{\hat{n}}{m}$ be the estimated load factor (in terms of the average number of flows that are mapped to the same index) on the array. Our simple estimator for n_1 is $\hat{n_1} = y_1 e^{\hat{\lambda}}$, where y_1 is the number of counters with value 1. This surprisingly simple estimator $\hat{n_1}$ turns out to be very accurate. In our experiments shown later, we observed an accuracy of $\pm 2\%$ using $\hat{n_1}$. Next, we explain the reasoning behind $\hat{n_1}$.

Since the order of packet or flow arrivals does not affect the final values in the counter array, we consider a hypothetical situation where all flows of size 2 and above were inserted in the counter array first. There are altogether $n - n_1$ of them. At this point, none of the flows of size 1 has been inserted. The number of flows hashed to an index can be modeled as a binomial distribution Binom($n - n_1$, $\frac{1}{m}$), which in turn

[6]We have carefully checked these parameters against the specifications in [12].

[7]Our experiments on other traffic traces exhibit similar distortion effects.

can be approximated by Poisson($\frac{n-n_1}{m}$). The total number of indices that are not hit by any flow at this point (i.e., indices where the counter value is 0) can be estimated as $m_0' \approx m \cdot e^{-\frac{n-n_1}{m}}$. Now, assume all the flows of size 1 are inserted into this array. Due to this insertion, some of these m_0' counters will become non-zero. The counters with value 1 will be those out of a total of m_0' that were zero before the insertion of n_1 flows of size 1, and were hit by exactly one of these new insertions. By the same argument as above, the total number of such indices is $m_0'\lambda_1 e^{-\lambda_1}$, where $\lambda_1 = \frac{n_1}{m}$. But this number should be equal to y_1. Therefore we have

$$y_1 = m_0'\lambda_1 e^{-\lambda_1} = m \cdot e^{-\frac{n-n_1}{m}} \cdot \frac{n_1}{m} \cdot e^{-\frac{n_1}{m}} = n_1 e^{-\frac{n}{m}}$$

which can be simplified as

$$n_1 = y_1 e^{\frac{n}{m}} \quad (2)$$

3.3 Estimating the flow distribution

One is tempted to generalize the above process to derive an estimator for the number of flows of size 2, 3, and so on (i.e., estimating n_2, n_3, ..., n_z). However, this proves to be difficult due to the following reason. While a counter of value 1 is definitely not involved in a collision, counter-values of 2 and above could be caused by the collision of two or more flows. For example, among the counters of value 2, some correspond to a flow of size 2, while the others could be two flows of size 1 hashing to the same index. Thus, while the estimate for n_1 (i.e., the number of flows of size 1) depends only on our estimate of n, the estimate of n_2 will depend on both n and n_1. More generally, the estimate of n_i will depend on our estimates of $n, n_1, n_2, \cdots, n_{i-1}$. Thus for a large flow size i, the estimate is more susceptible to errors due to this cumulative dependence effect, resulting in a sharp increase in estimation errors. Therefore, to accurately estimate flow distribution, we will take a more holistic approach, rather than estimating each quantity step by step. This approach, based on Expectation Maximization (EM) method for computing Maximum Likelihood Estimation (MLE), is the sole topic of the next section.

4. ESTIMATING FLOW DISTRIBUTION USING EXPECTATION MAXIMIZATION

In this section, we describe our Maximum Likelihood Estimation (MLE) algorithm that computes the flow distribution that is most likely to result in the observed counter values after the hash collisions. To find this MLE directly is difficult because there is neither a closed-form formula nor a computation procedure for $p(\phi|y)$, the distribution of the flow distribution ϕ conditioned on the observation y. The difficulty of computing $p(\phi|y)$ can be attributed to the fact that our observed data is incomplete.

To address this problem, we adopted a powerful method in statistics called Expectation Maximization (EM) to iteratively compute the *local*[8] MLE. EM is especially effective in finding MLE when the observation can be viewed as incomplete data. In our context, the observed counter values can be viewed as incomplete data, and the missing part is how flows collide with each other during hashing. The evaluation in Section 6 shows that our EM algorithm accurately estimates the flow distribution among all traces we have experimented with. To the best of our knowledge, this is the first work that applies EM algorithm to computing the MLE from a lossy data structure.

[8]EM algorithms in general can only guarantee to converge to a local maximum [15], while MLE often refers to the global maximum. With this understanding, we will omit the word local from subsequent discussions of MLE using EM.

4.1 Background on EM

Let y denote our observation and ϕ denote the random variable whose value we would like to estimate. In MLE, we would like to find out ϕ^* that maximizes $p(\phi|y)$. However, it is usually hard to compute such a ϕ^* because the formula $p(\phi|y)$ is either complicated or does not have a closed form due to missing data. The EM algorithm, which captures our intuition on handling missing data, works as follows. It starts with a guess of the parameters, and then replaces missing values by their expectations given the guessed parameters, and finally estimates the parameters assuming the missing data are equal to their estimated values. This new estimate of missing values gives us a better estimate of parameters. This process will be iterated multiple times until the estimated parameters converge to a set of values (typically a local maximum as mentioned above).

Formally, EM begins with a guess of the parameter ϕ^{ini}, which will serve as ϕ^{old} for the first iteration. Then the following two alternating steps will be executed iteratively.

Expectation step. $E_{old}(\log p(\gamma, \phi|y)) = \int (\log p(\gamma, \phi|y))$ $p(\gamma|\phi^{old}, y)d\gamma$, where the expectation averages over the conditional posterior distribution of the missing data γ, given the current estimate ϕ^{old}. We use the notation $Q(\phi, \phi^{old})$ to denote $E_{old}(\log p(\gamma, \phi|y))$ per the convention in statistics literature. For many applications, both $p(\gamma|\phi, y)$ and $p(\phi|\gamma, y)$ inside the integration formula above are straightforward to compute.

Maximization step. Let ϕ^{new} be the value of ϕ that maximizes $Q(\phi, \phi^{old})$. This ϕ^{new} will serve as ϕ^{old} for the next iteration.

These two steps will be iterated for a number steps until ϕ^{old} and ϕ^{new} are close enough to each other, a notion that will become rigorous later in Section 6.2.

4.2 Applying EM to our context

Our observation y, obtained from the output of online streaming module, is y_i ($i = 1, 2, ..., z$), the number of counters that have value i. Our goal is to estimate ϕ_i, the fraction of flows that are of size i ($i = 1, 2, ..., z$). Here z is the maximum counter value observed from the array.

Our EM algorithm for estimating ϕ is shown in Figure 4. We first need a guess of the flow distribution ϕ^{ini}, and the total number of flows n^{ini}. In our algorithm, we simply use the distribution obtained from the raw counter values as ϕ^{ini} and the total number of non-zero counters as n^{ini}. Based on this ϕ^{ini} and n^{ini}, we can compute, for each possible way of "splitting" an observed counter value, its average number of occurrences. Then the counts n_i for flows of corresponding sizes will be credited according to this average. For example, when the value of a counter is 3, there are three possible events that result in this observation: (i) $3 = 3$ (no hash collision); (ii) $3 = 1 + 2$ (a flow of size 1 colliding with a flow of size 2); and (iii) $3 = 1 + 1 + 1$ (three flows of size 1 hashed to the same index). Given a guess of the flow distribution, we can estimate the posterior probabilities of these three cases. Say the respective probabilities of these three events

Input: y_i, number of counters that have value i $(1 \le i \le z)$
Output: MLE for the flow distribution ϕ

1. Initialization: pick an initial flow distribution $\phi^{(ini)}$ and estimate the total flow count n^{ini} from Section 3.1.
2. $\phi^{new} := \phi^{ini}$; $n^{new} = n^{ini}$
3. **while** (convergence condition is not satisfied)
4. $\quad \phi^{old} := \phi^{new}$; $n^{old} := n^{new}$
5. $\quad$ **for** i :=1 **to** z
6. $\quad\quad$ **foreach** $\beta \in \Omega_i$
7. $\quad\quad\quad$ /*Ω_i is the set of all "collision patterns"
8. $\quad\quad\quad$ that add up to i, defined in Theorem 1*/
9. $\quad\quad\quad$ Suppose β is that f_1 flows of size s_1, f_2 flows of
10. $\quad\quad\quad$ size s_2, ..., and f_q flows of size s_q collide into
11. $\quad\quad\quad$ a counter of value i, then
12. $\quad\quad\quad$ **for** $j := 1$ **to** q
13. $\quad\quad\quad\quad$ $n_{s_j} := n_{s_j} + y_i * f_j * p(\beta|\phi^{old}, n, V = i)$
14. $\quad\quad\quad\quad$ /* Procedure for computing $p(\beta|\phi^{old}, n, V = i)$
15. $\quad\quad\quad\quad$ is shown in Theorem 1 and Lemma 1.*/
16. $\quad\quad\quad$ **end**
17. $\quad\quad$ **end**
18. $\quad$ **end**
19. $\quad n^{new} := \sum_{i=1}^{z} n_i$
20. $\quad$ **for** i:=1 **to** z
21. $\quad\quad \phi_i^{new} := n_i / n^{new}$
22. $\quad$ **end**
23. $\quad$ /* normalize the counts $n_i's$ into flow distribution ϕ*/.
24. **end**

Figure 4: EM algorithm for computing flow distribution

are 0.5, 0.3, and 0.2, and there are 1000 counters with value 3. Then we estimate that, on the average, 500, 300, and 200 counters split in the three above ways, respectively. So we credit 300 * 1 + 200 * 3 = 900 to n_1, the count of flows of size 1, and credit 300 and 500 to n_2 and n_3, respectively. Finally, after all observed counter values are split this way, we get the new counts n_1, n_2, ..., n_z, and obtain $n^{new} (= \sum_{i=1}^{z} n_i)$. We then renormalize them into a new (and refined) flow distribution ϕ^{new} . We will prove in Section 4.3 that this program is indeed an instance of the EM algorithm.

Computing the probability $p(\beta|\phi, n, v)$. Let both n and m (size of counter array) be very large so that we can approximate binomial distribution using Poisson. This approximation is necessary since our estimates of flow counts can be non-integers. Let λ_i denote the average number of size i flows (before collision) that are hashed to an (arbitrary) index in the array. In other words, $\lambda_i = \frac{n_i}{m} = \frac{n\phi_i}{m}$. We define $\lambda = \sum_{i=1}^{z} \lambda_i$, which is the average number of flows (of all sizes) that is hashed to an (arbitrary) index. Let *ind* be an arbitrary index into the array and v be the observed value at this index. Let β be the event that f_1 flows of size s_1, f_2 flows of size s_2, ..., f_q flows of size s_q collide into this slot, where $1 \le s_1 < s_2 < ... < s_q \le z$.

LEMMA 1. *Given ϕ and n, the a priori (i.e., before observing this value v) probability that event β happens is*

$$p(\beta|\phi, n) = e^{-\lambda} \prod_{i=1}^{q} \frac{\lambda_{s_i}^{f_i}}{f_i!}.$$

PROOF. Let B_i be the event that f_i flows of size s_i be mapped to the counter indexed by *ind*. Let C be the event that all other flows have zero arrivals to *ind*. Since the hashing is uniform, these events B_1, B_2, ..., B_q, and C are independent. Therefore, $p(\beta|\phi, n) = p(C|\phi, n) \prod_{i=1}^{q} p(B_i|\phi, n)$. Let $I = \{s_1, s_2, ..., s_q\}$. Then $p(B_i|\phi, n) = e^{-\lambda_{s_i}} \frac{\lambda_{s_i}^{f_i}}{f_i!}$ by Poisson approximation of binomial distribution. So, $\prod_{i=1}^{q} p\left(B_i|\phi, n\right) = \prod_{i=1}^{q} e^{-\lambda_{s_i}} \frac{\lambda_{s_i}^{f_i}}{f_i!} = \left(\prod_{j \in I} e^{-\lambda_j}\right) \left(\prod_{i=1}^{q} \frac{\lambda_{s_i}^{f_i}}{f_i!}\right)$
Also, $p\left(C|\phi, n\right) = \prod_{j \in \overline{I}} e^{-\lambda_j}$. Therefore,

$$p\left(\beta|\phi, n\right) = \left(\prod_{j \in \overline{I}} e^{-\lambda_j}\right) \left(\prod_{j \in I} e^{-\lambda_j}\right) \left(\prod_{i=1}^{q} \frac{\lambda_{s_i}^{f_i}}{f_i!}\right) = e^{-\lambda} \prod_{i=1}^{q} \frac{\lambda_{s_i}^{f_i}}{f_i!}$$

□

However, the situation changes after we have already seen v, the value at the counter indexed by *ind*.

THEOREM 1. *Let Ω_v be the set of all collision patterns that add up to v. Then $p(\beta|\phi, n, v) = \frac{p(\beta|\phi, n)}{\sum_{\alpha \in \Omega_v} p(\alpha|\phi, n)}$, where $p(\beta|\phi, n)$ and $p(\alpha|\phi, n)$ can be computed using Lemma 1.*

PROOF. Let Ω be the set of all possible collision patterns as defined before. Let us choose an arbitrary index *ind* and let V be the counter value at this index. By Bayes' rule,

$$p(\beta|\phi, n, V = v) = \frac{p(V = v|\beta, \phi, n) p(\beta|\phi, n)}{\sum_{\alpha \in \Omega} p(V = v|\alpha, \phi, n) p(\alpha|\phi, n)}$$

However, note that $p(V = v|\alpha, \phi, n) = 1$ for all $\alpha \in \Omega_v$ (including β) and $p(V = v|\alpha, \phi, n) = 0$ for all $\alpha \in \Omega - \Omega_v$. Therefore,

$$p(\beta|\phi, n, v) = \frac{p(\beta|\phi, n)}{\sum_{\alpha \in \Omega_v} p(\alpha|\phi, n)}$$

□

4.3 Our algorithm is an EM algorithm

We next prove that the algorithm shown in Figure 4 is indeed an EM algorithm. This proof is important since the fact that the algorithm is an instance of EM guarantees that the outputs from the iterations of the algorithm will converge to a set of local MLEs, according to [15].

THEOREM 2. *The algorithm in Figure 4 is an EM algorithm.*

PROOF. Let γ_{ij} denote the number of size i flows that are collided (merged) into counters of value j $(1 \le i \le j \le z)$. These are the missing data that the algorithm in Figure 4 needs to guess in order to estimate the flow distribution ϕ. The complete data likelihood function $L(\phi)$ (i.e., $p(\gamma, \phi|y)$ defined in Section 4.1), assuming γ_{ij} is known, is $\sum_{i=1}^{z} \sum_{j=i}^{z} \gamma_{ij} \log \phi_i$.

Then in the expectation step,

$$E_{(\phi^{old}, n)}[L(\phi)|y] = \sum_{i=1}^{z} \sum_{j=i}^{z} E[\gamma_{ij}|\phi^{old}, y, n] \log \phi_i$$

This corresponds to $Q(\phi, \phi^{old})$ in Section 4.1. Let $\gamma_i = \sum_{j=i}^{z} \gamma_{ij}$. Define $n_{ij} = E[\gamma_{ij}|\phi^{old}, y, n]$ and $n_i = E[\gamma_i|\phi^{old}, y, n]$. By the linearity of expectation, we know that $n_i = \sum_{j=i}^{z} n_{ij}$. Therefore, $E_{(\phi^{old}, n)}[L(\phi)|y] = \sum_{i=1}^{z} n_i \log \phi_i$. Note that the definition of n_i here matches the computation of n_i in our algorithm (lines 5 to 18).

Finally in the maximization step, we need to maximize $\sum_{i=1}^{z} n_i \log \phi_i$, subject to the constraint $\sum_{i=1}^{z} \phi_i = 1$. Here

n_i $(i = 1, 2, ..., z)$ are constants and $\phi_i's$ are the variables. Using the method of Lagrange multiplier, we know that the maximum value is achieved when $\phi_i = \frac{n_i}{\sum_{j=1}^{z} n_j}$. This is exactly the renormalization step in our program (lines 19 to 23) shown in Figure 4. Therefore, our algorithm is indeed an EM algorithm. □

4.4 Computational complexity of the EM algorithm.

It is easy to enumerate all possible events that give rise to a small counter value. But, for large counter values, the number of possible events (hash collisions) that could give rise to the observed value is immense. Thus it is not possible to exhaustively compute the probabilities for all such events. The "Zipfian" nature of flow-size distribution comes to our rescue here. To reduce the complexity of enumerating all events that could give rise to a large counter value (say larger than 300), we ignore the cases involving the collision of 4 or more flows at the corresponding index. Since the number of counters with a value larger than 300 is quite small, and collisions involving 4 or more flows occur with a low probability, this assumption has very little impact on the overall estimation mechanism. With similar justifications we ignore events involving 5 or more collisions for counters larger than 50 but smaller than 300 and those involving 7 or more collisions for all other counters. This reduces the asymptotic computational complexity of "splitting" a counter-value j to $O(j^3)$ (for $j > 300$). Note that we need to do this computation only once for *all* counters that have a value j, and the number of unique counter-values is quite small (as discussed earlier in Section 2.3).

Finally, since the numbers of counters with very large values (say larger than 1000) is extremely small, we can ignore splitting such counter values entirely and instead report the counter value as the size of a single flow. This will clearly lead to a slight overestimation of the size of such large flows, but since the average flow size (≈ 10) is two to three orders of magnitude smaller than these large flows, this error is minuscule in relative terms.

These optimizations bring the overall computational complexity well under control. On a 3.2 GHz Intel Pentium 4 desktop, each iteration of the EM takes about 20 seconds. If the measurement epoch is 100 seconds long and we terminate the estimation after five iterations, then the estimation can run as fast as the data streaming module.

5. MULTI-RESOLUTION ESTIMATION OF FLOW DISTRIBUTION

As shown in Figure 3, the raw counter value distribution deviates more and more from the actual flow distribution as the size of the counter array decreases. Our experiments in Section 6 show that the accuracy of estimation falls sharply if the size of the array is less than $\frac{2}{3}$ of the total number of flows n. Therefore, for the accurate estimation of flow distribution, we need a counter array that contains at least $\frac{2}{3}n$ entries. However, in real-world Internet traffic, the number of flows in the worst case can be many times more than in the average case. Provisioning enough counters for the worst case would result in excessive waste of precious SRAM in the average case. In this section, we present a multi-resolution version of our solution that uses a fixed-size array of counters, and allows a graceful degradation in estimation accuracy when the total number of flows increases. This makes the scheme accurate and memory-efficient for the average case while its accuracy degrades only slightly for the worst case. Our design is inspired by a multi-resolution scheme used in [6]. We apply it here to a different context.

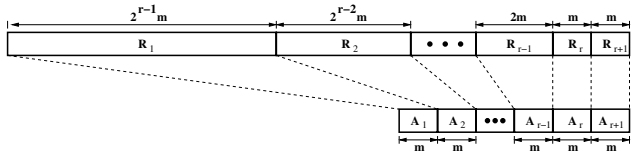


Figure 5: The Multi-Resolution Array of Counters.

1. **Initialize**
2. $r = \log_2(M/m)$
3. $R_i = \left\{ \begin{array}{cc} \left[(1-\frac{1}{2^{i-1}})M, (1-\frac{1}{2^i})M\right) & i = 1, 2, 3, ..., r \\ \left[(1-\frac{1}{2^r})M, M\right) & i = r+1 \end{array} \right\}$
4. Arrays A_1, A_2,$\cdots$, A_{r+1} are all initialized to 0

5. **Update**
6. Upon the arrival of a packet pkt
7. $ind :=$ hash($pkt.flow\_label$);
8. if ($ind \in R_j$)
9. $A_j[ind \bmod m]$++;

Figure 6: Algorithm for updating MRAC.

Our *Multi-Resolution Array of Counters (MRAC)* scheme works as follows. Imagine a virtual array of counters that is large enough to accurately estimate the flow distribution even in the worst case. However, the physical (actual) counter array size is much smaller. Therefore, the virtual array needs to be mapped/folded to the actual physical array as shown in Figure 5. Here we describe a base-2 version of our mapping. Its generalization to any arbitrary base 'b' is straightforward. In the base-2 version, we map a logical array of $M = 2^r m$ counters to $r + 1$ physical arrays of size m each. Half of the hash space will be mapped to (folded into) array 1, half of the remaining hash space (i.e., $\frac{1}{4}$ of the total hash space) will be mapped to array 2, and so on. Finally, we are left with two blocks of hash space of size m each. They are directly mapped to arrays r and $(r+1)$. The total space taken by the arrays is $m(\log_2 \frac{M}{m} + 1)$.

This actual mapping/folding algorithm is shown in Figure 6. As described above, the arrays $A_1, A_2, ..., A_r, A_{r+1}$ cover the respective hash ranges of $[0, \frac{1}{2}M)$, $[\frac{1}{2}M, \frac{3}{4}M)$, $[\frac{3}{4}M, \frac{7}{8}M)$, $\cdots$, $[(1-\frac{1}{2^{r-1}})M, (1-\frac{1}{2^r})M)$, $[(1-\frac{1}{2^r})M, M)$. If a hash index ind is mapped to a array, the counter indexed by ($ind \mod m$) in that array will be incremented. Therefore, the values of 2^{r-1} counters in the virtual array map to (fold into) 1 counter in array A_1, and the values of 2^{r-2} virtual counters map to 1 counter in array A_2, and so on. The $(r + 1)$ arrays together cover the entire virtual hash space. The regions covered by any two arrays are disjoint.

Such a mapping is implicitly a flow sampling (not packet sampling) scheme. Array A_1 processes approximately $\frac{1}{2}$ of the flows (i.e., every packet in approximately half of the flows), array A_2 processes approximately $\frac{1}{4}$ of the flows, and so on. Note that the computational complexity of this scheme is almost the same as the baseline approach, which

is one hash function computation and one memory access to SRAM. The only additional processing here is to recognize the range that a hash value falls into and to perform a modulo operation ("*ind* mod m" in line 9 of Figure 6). Since all operations involve 2's powers, they can be implemented efficiently using simple binary logic.

The estimation algorithm works as follows. It first picks an array that will result in the best estimate of the original flow distribution. The criteria of picking such an array will be discussed next. Suppose the array we pick is 2^{-i} of the size of the virtual array, that is, this array samples approximately 2^{-i} fraction of the flows. The algorithm first estimates the flow distribution from the array using the baseline approach described in the previous two sections, and then scales the result by 2^i to obtain the estimate for the overall traffic. Since the number of very large flows (say larger than 1000 packets) is quite small, we can use the counter values larger than 1000 from all resolutions to refine our estimation for the tail of the distribution. For each of these large counter values, we subtract the average counter value in the corresponding resolution and use the result as the estimated size of the large flows hashed to this counter.

In general, the arrays where the sampling rate is high (i.e., the arrays that cover large portions of the virtual hash space) tend to be "over-crowded" (i.e., with higher average number of flows mapped to the same slot). This corresponds to using a very small array of counters, which results in inaccurate estimation. On the other hand, when the sampling rate is low (i.e., when the array covers a very small portion of the virtual hash space), the estimation from the corresponding array will be accurate, but the errors due to (flow) sampling become high. Therefore, there is a clear tradeoff between the loss of accuracy due to "over-crowding" on the one hand and due to sampling on the other. We find that there exists an optimal array size in the middle that minimizes the overall loss of accuracy, which can be found using the following criteria. We pick an array with as high sampling rate as possible, under the constraint that no more than 1.5 flows are mapped to the same slot on the average. The reasoning behind this rule is similar to that used in two existing multi-resolution based schemes [6, 16] (for different applications). We omit the details here in the interest of space.

Finally, the above design with base-2 can be generalized to any arbitrary base. Choosing a base that is a power of 2 allows efficient hardware and software implementation. Our implementation evaluated in Section 6 uses base-4. The base-4 algorithm needs 50% less memory than base-2, with nominal loss in estimation accuracy.

6. EVALUATION

In this section, we evaluate the accuracy of our estimation mechanism using real-world Internet traffic traces. We also compare our results with those obtained in [1] from sampled traffic. Our experiments demonstrate that our mechanism achieves very high accuracy, which is typically an order of magnitude better than sampling-based approaches.

6.1 Traffic traces

We use three sets of traces in our evaluation. The first set comprises of two packet header traces obtained from a tier-1 ISP backbone, collected by a Gigascope server [17] on a high speed link leaving a data center in October, 2003. Each of the packet header traces lasts a few hours, consists of ∼700 million packet headers and carries ∼350 GB traffic. In our experiments, we used segments taken from these two traces, one for heavier traffic load on a weekday and the other for light traffic load at a weekend. Table 1 lists the number of flows and packets in each trace.

Source	Trace	# of flows	# of packets
ISP	Weekday	11,341,289	68,595,755
	Weekend	1,239,746	8,861,457
NLANR	Long	563,080	1,769,431
	Medium	192,380	2,668,269
	Short	55,515	158,243
[1]	CAMPUS	425,702	10,065,600
	COS	6,038,554	37,000,000
	PEERING	1,289,825	10,000,000

Table 1: Traces used in our evaluation.

The second set of traces we used are publicly available traffic traces from NLANR. We use three NLANR traces[9] named "Long", "Medium", and "Short", based on the number of flows in each trace (Table 1). Notice that the trace "Long" actually has fewer packets than "Medium". However, the attribute of significance in our evaluation is the number of flows in each trace, and the names are intuitive in this light.

Finally, we use a set of three traces from [1] to compare with previous work on estimating flow-distribution from sampled statistics. Trace "CAMPUS" was collected at a LAN near the border of a campus network during a period of 300 minutes. Trace "COS" was collected at an OC3 link at Colorado State University during January 25 and 26, 2003. This period overlaps the onset of Slammer worm [18]. Trace "PEERING" was collected at a peering link for a period of 37 minutes.

6.2 Evaluation metrics

For comparing the estimated flow distribution with the actual distribution, we considered two possible metrics, *Mean Relative Difference (MRD)* and *Weighted Mean Relative Difference (WMRD)*. We eventually adopt WMRD as our evaluation metric. The rationale for this choice is given below.

The metric MRD is often used in measuring the distance between two probability distributions or mass functions, defined in our context as follows. Suppose the number of flows of size i is n_i and our estimate of this number is $\hat{n_i}$. The relative error in estimation (i.e., relative difference) is given by $|n_i - \hat{n_i}|/(\frac{n_i+\hat{n_i}}{2})$. The mean relative difference over all flow sizes is obtained by taking the mean of relative difference over all possible flow sizes 1, 2, 3, ..., z. Therefore, the MRD between the estimated and actual distribution is given by:

$$MRD = \frac{1}{z}\sum_i \frac{|n_i - \hat{n_i}|}{\left(\frac{n_i+\hat{n_i}}{2}\right)}$$

However, this metric is not suitable for estimating flow distribution for the following reason. The "Zipfian" nature of the Internet traffic implies that there are a large number of small flows and only a few large flows. In other words, when i becomes larger, n_i becomes smaller, and $|n_i - \hat{n_i}|/(\frac{n_i+\hat{n_i}}{2})$ becomes larger. Therefore, the errors in estimating the tail

[9]We experimented on many other NLANR traces, which yield similar results as reported in this paper.

of the distribution (i.e., $|n_i - \hat{n_i}|/(\frac{n_i+\hat{n_i}}{2})$ for large values of i) dominate the value of MRD. This makes no sense since the main body of the distribution is the large number of small flows, the estimation accuracy of which is discounted in MRD.

To reflect the errors in estimating the number of large and small flows in proportion to their actual population, we adopt the aforementioned second metric called *Weighted Mean Relative Difference (WMRD)*. It is proposed and used in [1], for the same purpose of evaluating the accuracy of estimated flow distribution. In WMRD, we assign a weight of $\frac{n_i+\hat{n_i}}{2}$ to the relative error in estimating the number of flows of size i. Thus the value of WMRD is given by:

$$WMRD = \frac{\sum_i \frac{|n_i-\hat{n_i}|}{\left(\frac{n_i+\hat{n_i}}{2}\right)} \times \frac{n_i+\hat{n_i}}{2}}{\sum_i \left(\frac{n_i+\hat{n_i}}{2}\right)} = \frac{\sum_i |n_i-\hat{n_i}|}{\sum_i \left(\frac{n_i+\hat{n_i}}{2}\right)}$$

WMRD is also used in our EM algorithm to determine how close our estimate is to the convergence point. In our algorithm, we choose a threshold ϵ, and terminate our iterative estimation procedure when the WMRD of the estimates produced by two consecutive estimates falls below ϵ. The intuition here is that, as estimates get closer to the convergence point, the improvement from one iteration to the next becomes smaller, implying a smaller WMRD between the estimates produced by two consecutive estimates.

6.3 Bucketing flow distribution

As can be seen from Figure 3, the tail of the flow distribution plot is very noisy. This is due to the fact that a small number of large flows are distributed in a large size range in a very sparse way. To obtain a more intuitive visual depiction of the flow distribution, we use a *bucketing* scheme to smooth out the noise. Buckets are sets of one or more consecutive integers. The total number of flows in a bucket is the sum of the number of flows of each unique size in the bucket. For small flow sizes, where there are a large number of flows for each unique size, we use a bucket size of 1, implying no smoothing. As we proceed towards large flow sizes, gaps between two flow sizes that have non-zero counts start appearing (and then widening). We scale the bucket size appropriately so that most buckets have at least one flow. In the figures depicting flow distribution, each bucket is depicted as a data point, with the mid-point of the bucket as its x-coordinate and the total number of flows in the bucket divided by the bucket size as its y-coordinate. *We emphasize that this bucketing scheme is used only for better visualization of results. Our estimation mechanism and numerical results (in WMRD) reported later in this section do not use smoothing of any form.*

6.4 Estimation using array of counters

As mentioned earlier in Section 5, we show that the estimation procedure is likely to be more accurate when the number of counters is close to or larger than the total number of flows in the measurement epoch. Table 2 shows the effect of the choice of number of counters over estimation accuracy for the NLANR traces. The deviation of both the initial guess (taken from the observed counter value distribution) and the final estimate after 20 iterations of the EM algorithm becomes larger when the number of counters become smaller. However, this increase in WMRD is very small when the number of counters stay larger than or equal to the number of flows. The increase becomes pronounced only after the number of counters drops to less than $\frac{2}{3}$ of the number of flows.

Trace	# of flows in trace	Array size	WMRD of raw data	WMRD of final estimate
Long	563,080	1024K	0.38195	0.00643
		512K	0.70140	0.02664
		256K	1.13521	0.25858
		128K	1.59242	0.95548
Medium	192,380	512K	0.23010	0.01715
		256K	0.43337	0.03424
		128K	0.75778	0.10895
		64K	1.19023	0.42463
Short	55,515	128K	0.31478	0.01138
		64K	0.59331	0.01929
		32K	1.01238	0.14560
		16K	1.48354	0.66332

Table 2: WMRD of initial guesses and final estimates.

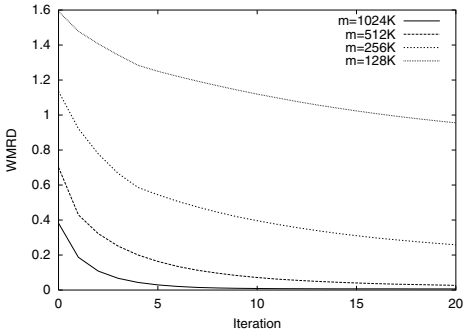


Figure 7: The WMRD of the estimate vs. the number of iterations of the estimation algorithm, for the trace "Long".

Figure 7 shows how the WMRD of the estimates decreases when the number of EM iterations increases. The trace used for this experiment is trace "Long" containing 563,080 flows. Each curve corresponds to a different choice of the number of counters, ranging from 128K (2^{17}) to 1M (2^{20}). The points for iteration 0 correspond to the WMRD of the initial guess, obtained from the distribution of the raw counter values. All the curves show a downward trend on WMRD to approach zero, indicating progress toward convergence. The curves for $m = 1024K$ and $m = 512K$ begin with much better initial guesses, thus achieving a much smaller WMRD (in absolute value) within a small number of iterations. This reinforces the notion that using approximately the same number of counters as the number of flows provides much better estimation accuracy than using a significantly smaller number of counters. We observe similar results on other traces.

Figure 8 presents the results of running our estimation mechanism on the trace "Long" (similar results are observed on all traces in Table 2). In this experiment, the number of counters m was set to a 2's power that is closest to the number of flows n. Each figure has three curves, corresponding to the actual distribution of flow sizes, the distribution of raw counter values, and the result of our estimation mechanism, respectively. The near overlap of our estimate with

the actual distribution indicates the high accuracy of the estimation.

6.5 Comparison with sampling-based estimation

We compare the accuracy of our mechanism against the best known mechanism [1] for estimating flow distribution from sampled traffic. It should be noted that this comparison is not to point out any shortcomings of [1]. Indeed, we believe that the solution in [1] is close to the best that can be done on the sampled data. The gain of accuracy in our mechanism comes from the highly efficient online data streaming algorithm, which saves us from having to skip 90% or 99% of traffic as done in the sampling-based approach.

The comparison is performed on the same data used in [1] for best fairness. Figure 9 shows the performance of our mechanism, as well as that of [1] on two sub-traces containing Web and DNS traffic, extracted from the trace "COS". There are four curves in each figure. Two of them correspond to the actual flow distribution and estimation from our mechanism (using similar number of counters as flows, as described before) respectively. The other two curves are plotted using the data corresponding to the flow distribution estimated from sampled traffic [1], with sampling rates of 10% and 1% respectively. The accuracy of our mechanism is highlighted by the nearly complete overlap between the estimate curve and the actual flow-size distribution curve. This perfect overlap actually makes the four curves look like three curves in each figure. Estimation based on sampled traffic, on the other hand, deviates much more from the actual flow distribution.

6.6 Estimation of flows of size 1

As mentioned in Section 3.2, the number of flows of size 1 can be very accurately estimated using the estimator $\hat{n_1} = y_1 e^{\frac{\hat{n}}{m}}$. Table 3 lists the estimated values of the number of flows of size 1 vs. the actual values for three different NLANR traces. In all cases, the estimates were within 2% of the actual value, thus demonstrating the high accuracy of the estimator.

Table 4 measures the effectiveness of our mechanism in detecting sharp changes in the number of flows of size 1 compared with sampling-based approach. The trace "COS" contains a large number of single-packet UDP flows sent to random destination IP addresses by hosts infected with MS SQL server worm. For evaluating the mechanisms on their abilities to estimate the number of flows of size 1, a sanitized version of the trace, with all the worm packets removed, was also processed. Table 4 compares results from the sampling-based approach [1] with estimates using our mechanism (sampling rate[10] is 0.001). We can see that the sampling-based approach reports an increase of 20% (19,433 to 24,275) in the number of flows of size 1, while going from the sanitized trace to the complete trace. However, the actual increase (451,489 to 5,059,379) is close to 95%. This indicates the difficulty of detecting sharp changes in the number of flows of size 1 using sampling. The estimates from our mechanism, on the other hand, are quite accurate, and closely reflect the change (from 453,703 to 5,025,940).

[10]The same sampling rate is used in [1] in the same context. Similar sampling rates are also adopted by large commercial ISPs due to the high volume of Internet traffic.

Trace	# of flows of size 1	Estimated value
Long	365,265	357,956
Medium	80,459	79,108
Short	37,073	36,844

Table 3: Estimation of the number of flows of size 1.

Trace	Flows of size 1	Flows of size 1 in sampled data set	Estimated value
Original	5,059,379	24,275	5,025,940
Worm-excluded	451,489	19,433	453,703

Table 4: Detecting changes in the number of flows of size 1 in trace "COS".

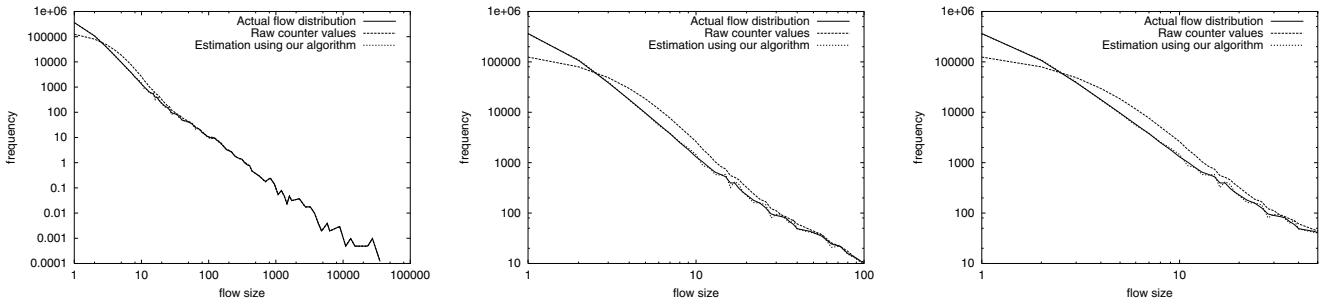
6.7 Estimation using MRAC

In this section, we present the results of estimating the flow distribution using Multi-Resolution Array of Counters (MRAC). The following base-4 configuration is used in all experiments in the sequel. The MRAC consists of three *virtual* arrays of logical range 87,382 (64K*4/3), 349,525 (64K*16/3) and 1,048,576 (64K*16). All these virtual arrays are implemented using a single hash function with range 0 to $2^{20}-1$ (=1,048,575). Each physical array has 64K counters, and the total space requirement of this MRAC configuration is 192K counters. Note that this size can be much smaller than the total number of flows in the three traces we will experiment on, which ranges from 55K to 563K.

Figure 10 shows our estimation results on three traces of different sizes. Except for some local fluctuations, which can be attributed to information loss due to sampling, the estimates are very close to the actual distribution most of the time, as reflected by near overlaps of the curves. In all these estimates, the most accurate resolution (virtual array) is determined automatically using the mechanism described in Section 5 (without using the undue knowledge about the actual number of flows). The WMRD values of the three traces in Figure 10 are 0.08557, 0.05001, and 0.03911, respectively. Note that although the WMRD values are worse than obtainable from a single counter array of suitable size, multi-resolution schemes saves considerable amount of SRAM (e.g., the "Long" trace would require 512K counters in the single resolution scheme). Moreover, MRAC's accuracy is still better than the best estimation accuracy obtainable using sampling based approaches (WMRD around 0.1 in the best cases as shown in [1]).

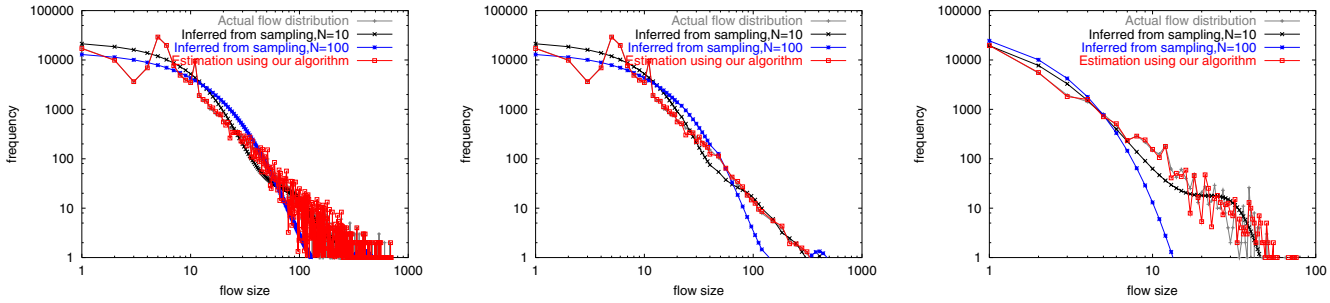
7. RELATED WORK

Previous work on estimating the flow distribution has mostly focused on inferring it from sampled traffic. In [4], the authors studied the statistical properties of packet-level sampling using real-world Internet traffic traces. This is followed by [1] in which the flow distribution is inferred from the sampled statistics. After showing that the naive scaling of the flow distribution estimated from the sampled traffic is in general not accurate, the authors propose an EM algorithm to iteratively compute a more accurate estimation. EM is also used in this paper, but in a very different way. Here we try to use EM to compensate for the information loss due to hash collisions while they use EM to compensate



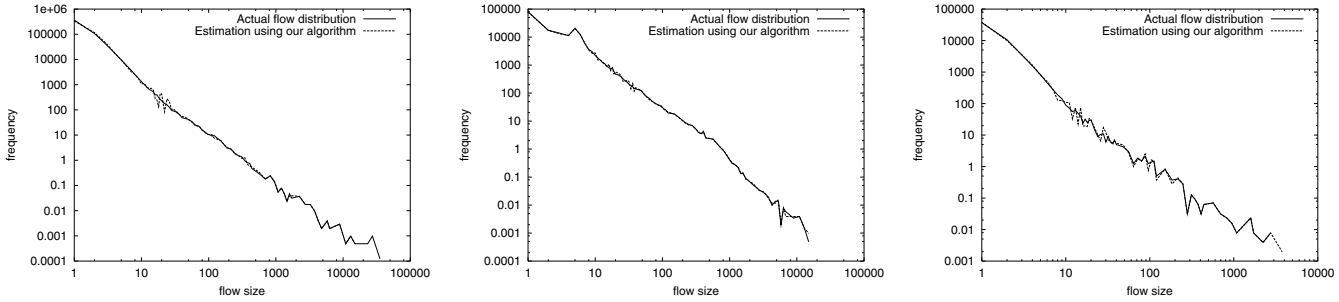
(a) Complete range of flow-sizes. (b) Zoomed-in on small flow sizes (c) Zoomed-in further.

Figure 8: Actual, raw, and estimated distributions for the trace "Long".



(a) Sampling vs. array of counters – Web traffic. (b) Replot with bucket-based smoothing – Web traffic. (c) Sampling vs. array of counters – DNS traffic.

Figure 9: Comparison of sampling and estimation based on array of counters.



(a) Trace "Long" (563,080 flows). (b) Trace "Medium" (192,380 flows). (c) Trace "Short" (55,515 flows).

Figure 10: Original and estimated distributions using MRAC.

for the information loss due to the low sampling rate (10% or 1%). Our algorithm is much more accurate because the information loss due to hash collisions is much less than due sampling that skips 90% to 99% of packets.

Recent work [2] discusses the inaccuracy of estimating flow distribution from sampled traffic, when the sampling is performed at the packet level. The authors also find that sampling at the flow level leads to more accurate estimations. The study in [2] is focused mostly on the theoretical aspects of sampling. Although they suggest that Bloom filters or bitmap algorithms can be used for flow-based sampling, no concrete mechanism is proposed. The multi-resolution version of our data structure uses sampling at the flow level, which is in line with the findings of [2].

Counter-arrays have been used in a number of systems and applications within the area of network measurement and

monitoring. For example, they have been the building blocks for detecting traffic changes [19], identifying large flows [5], and constructing Bloom filters [20] that allow both insertions and deletions (called counting Bloom filter) [21]. To the best of our knowledge, there is no prior work in the direction of "inverting" the hash collision process in the counter array using Bayesian statistics.

8. CONCLUSION

Estimating the distribution of flow sizes is important in a number of network applications. Current solutions rely on extrapolating the distribution learned from packet-level sampling. Although sophisticated methods have been developed for this, the loss of information due to packet sampling ultimately restricts the accuracy of any estimates recovered from it. We propose a novel data streaming scheme that achieves much more accurate estimation than sampling-based approaches. The scheme is based on a very simple data structure – an array of counters. Due to this simplicity, the scheme is able to operate at very high link speed (e.g., 40 Gbps) using a small amount of SRAM. However, since our scheme does not have enough space and time to resolve hash collisions, our observation from the counter arrays is a highly distorted version of the flow distribution. We develop a sophisticated mechanism based on expectation maximization to invert this distortion. We evaluate our mechanism on multiple Internet traffic traces, including the traces obtained from a tier-1 ISP's backbone network, and publicly available traces from NLANR. The experimental results demonstrate that our scheme achieves an order of magnitude better accuracy than sampling based approaches. We also develop a multi-resolution version of the scheme that achieves a graceful degradation in estimation accuracy when the number of flows is much larger than the size of counter array. This allows us to provision memory resources for the average case, while losing estimation accuracy only slightly in the worst case. Our algorithm not only dramatically improves the accuracy of flow distribution measurement, but also contributes to the field of data streaming by formalizing a methodology and applying it to a new context.

9. ACKNOWLEDGMENTS

This work is supported in part by NSF ITR Grant ANI-0113933 and NSF CAREER Award ANI-0238315. We would like to thank Dr. Nick Duffield for generously giving us access to the Internet traffic traces and results from his work on sampling-based estimation[1], which enabled our comparison of the two approaches. We also thank Dr. Oliver Spatscheck for providing us the Internet packet header traces collected by Gigascope servers. Finally, we thank the anonymous reviewers whose insightful comments have helped improve the quality of this paper.

10. REFERENCES

[1] N. Duffield, C. Lund, and M. Thorup, "Estimating flow distributions from sampled flow statistics," in *Proc. ACM SIGCOMM*, Aug. 2003.

[2] N. Hohn and D. Veitch, "Inverting sampled traffic," in *Proc. ACM SIGCOMM Internet Measurement Conference*, Oct. 2003.

[3] N. Duffield, C. Lund, and M. Thorup, "Charging from sampled network usage," in *Proc. ACM SIGCOMM Internet Measurement Workshop*, Nov. 2001.

[4] N. Duffield, C. Lund, and M. Thorup, "Properties and prediction of flow statistics from sampled packet streams," in *Proc. ACM SIGCOMM Internet Measurement Workshop*, Nov. 2002.

[5] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proc. ACM SIGCOMM*, Aug. 2002.

[6] C. Estan and G. Varghese, "Bitmap algorithms for counting active flows on high speed links," in *Proc. ACM SIGCOMM Internet Measurement Conference*, Oct. 2003.

[7] A. Medina, N. Taft, K. Salamatian, and S. Bhattacharyyaand C. Diot, "Traffic matrix estimation: Existing techniques and new directions," in *Proc. ACM SIGCOMM*, Aug. 2002.

[8] S. Vaton and A. Gravey, "Iterative bayesian estimation of network traffic matrices in the case of bursty flows," in *Proc. ACM SIGCOMM Internet Measurement Workshop*, Nov. 2002.

[9] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg, "Fast accurate computation of large-scale IP traffic matrices from link loads," in *Proc. ACM SIGMETRICS*, June 2003.

[10] Y. Zhang, M. Roughan, C. Lund, and D. Donoho, "An information-theoretic approach to traffic matrix estimation," in *Proc. ACM SIGCOMM*, Aug. 2003.

[11] S. Muthukrishnan, "Data streams: Algorithms and applications," available at http://athos.rutgers.edu/ muthu/.

[12] S. Ramabhadran and G. Varghese, "Efficient implementation of a statistics counter architecture," in *Proc. ACM SIGMETRICS*, 2003.

[13] M. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient hardware hashing functions for high performance computers," *IEEE Trans. on Computers*, vol. 46, no. 12, pp. 1378–1381, Dec. 1997.

[14] K. Whang, B. Vander-Zanden, and H. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Transactions on Database Systems*, 1990.

[15] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society, Series B*, vol. 39, no. 1, pp. 1–38, 1977.

[16] A. Kumar, J. Xu, J. Wang, O. Spatschek, and L. Li, "Space-Code Bloom Filter for Efficient per-flow Traffic Measurement," in *Proc. IEEE Infocom*, Mar. 2004.

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

[18] D. Moor, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "The spread of the sapphire/slammer worm," in *Technical Report, CAIDA*, 2003.

[19] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: methods, evaluation, and applications," in *Proc. ACM SIGCOMM Internet Measurement Conference*, Oct. 2003.

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

[21] 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.