A Data Streaming Algorithm for Estimating Subpopulation Flow Size Distribution[*]

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

ABSTRACT

Statistical information about the flow sizes in the traffic passing through a network link helps a network operator to characterize network resource usage, infer traffic demands, detect traffic anomalies, and improve network performance through traffic engineering. Previous work on estimating the flow size distribution for the *complete population* of flows has produced techniques that either make inferences from sampled network traffic, or use data streaming approaches. In this work, we identify and solve a more challenging problem of estimating the size distribution and other statistical information about *arbitrary subpopulations* of flows. Inferring subpopulation flow statistics is more challenging than the complete population counterpart, since subpopulations of interest are often specified *a posteriori* (i.e., after the data collection is done), making it impossible for the data collection module to "plan in advance".

Our solution consists of a novel mechanism that combines data streaming with traditional packet sampling to provide highly accurate estimates of subpopulation flow statistics. The algorithm employs two data collection modules operating in parallel — a NetFlow-like packet sampler and a streaming data structure made up of an array of counters. Combining the data collected by these two modules, our estimation algorithm uses a statistical estimation procedure that correlates and decodes the outputs (observations) from both data collection modules to obtain flow statistics for any arbitrary subpopulation. Evaluations of this algorithm on real-world Internet traffic traces demonstrate its high measurement accuracy.

Categories and Subject Descriptors

C.2.3 [**Computer Systems Organization**]: Computer-Communication Networks: Network Operations—*Network Monitoring*; E.1 [**Data**]: Data Structures—*Arrays*

[*]This work was supported in part by NSF CAREER award ANI-0238315.

General Terms

Algorithms, Measurement

Keywords

Traffic Analysis, Flow Statistics, Statistical Inference, EM Algorithm, Data Streaming

1. INTRODUCTION

Monitoring the aggregate IP traffic passing through a high-speed link is a complicated task. *Flows* provide a convenient abstraction, allowing aggregation of the packet-level information into coarser-granularity flow-level information. Flow statistics, such as the total number of flows and the distribution of flow sizes, capture significant properties of the underlying network traffic, and are of interest to both service providers and client network operators.

The Flow Size Distribution (FSD) is by far the most fundamental statistic since its accurate estimation can enable the inference of many other statistics such as the total number of flows and the average flow size. However, there are a number of applications where it would be useful to measure the flow size distribution of a particular *subpopulation*, i.e., a subset of the complete flow population. Such subpopulations may be defined by a traffic subtype (e.g., web or peer-to-peer traffic), by a source or destination IP address/prefix, by protocol (e.g., TCP or UDP), or some combination thereof. For example, to investigate a sudden slowdown in DNS lookups, a network operator may specify all flows with source or destination port 53 as the subpopulation of interest and query the data collected in the preceding hour.

It would be straightforward to estimate such distributions if per-flow state were maintained at routers. However, maintaining per-flow state at line speeds of 10 Gbps (OC 192) and above, is prohibitively expensive. Consequently, a significant amount of research effort has been devoted to estimating the FSD [1, 2, 3] from incomplete data, such as packet samples. The most accurate solution for estimating the FSD only allows the estimation of this distribution for the complete population of flows at the measurement point [3].

The ability to monitor the SubFSD for any arbitrary subpopulation provides a powerful and versatile tool for the characterization of traffic traversing a monitoring point. Since FSD is a special case of SubFSD, the motivations for estimating the FSD, ranging from accounting to anomaly detection [1, 2, 3], also apply to SubFSD. In addition, the capability to "zoom-in" and take a closer look at any subpopulation, supports a number of exciting new applications. We highlight examples in two areas:

Security. DDoS attacks and worm propagation often manifest themselves as anomalous deviations in the flow distribution, such as a sudden increase in the number of flows with exactly one packet [1]. Such deviations may not be statistically significant in the aggregate traffic, but may be clearly discernible if appropriate subpopulations are investigated. For example, in a subnet containing a DNS server and a web server, the flows of size 1 due to DNS traffic might mask out a spurt in flows of size 1 due to a SYN-flooding DDoS attack against the web server. On the other hand, looking at the SubFSD of HTTP flows will immediately reveal the attack.

Traffic Engineering. Knowledge of the SubFSD of various subpopulations within the aggregate traffic can enable fine-grained traffic engineering. For example, the impact of routing disruptions is often limited to specific subpopulations of traffic between the Origin-Destination (OD) pairs affected by the disruption. When only statistics about the complete flow population is available, monitoring this impact may prove to be a fairly involved problem. On the other hand, with the ability to monitor arbitrary subpopulations, this impact can be easily monitored by estimating the SubFSD for the subpopulations affected by the disruption. Note again that in this example, the subpopulation of interest is not known till after the event of interest has occurred.

Recent research has also produced mechanisms for estimating other important statistics such as the total number of flows [4], the number of flows of size 1 [1, 3], and the first and second moments of the FSD [5]. However, such solutions are often restricted to producing estimates for only the complete population of flows. Since estimates of the SubFSD can be used to infer these statistics for the corresponding subpopulation, a valuable by-product of our solution is the new ability to estimate such derived statistics for arbitrary subpopulation.

The difficulty of estimating the Subpopulation Flow Size Distribution (SubFSD) is twofold. First, the subpopulation of interest is typically not known in advance, i.e., before the act of data collection. Second, the number of subpopulations that could potentially be interesting is immense. These two requirements imply that any solution to the problem has to be generic enough to allow for the estimation of the FSD for any arbitrary subpopulation which is unknown during the act of data collection from live traffic.

In this work, we design an efficient mechanism to provide accurate estimates of the subpopulation flow size distribution for arbitrary subpopulations specified after the act of data collection. Our solution requires the system to maintain two online data structures, proposed and used in earlier work in this context, but only in isolation from one another. One is a variant of sampled NetFlow [6], already running on most routers. The other is a data structure used previously in a data streaming algorithm for estimating the FSD of the complete population [3]. We propose an estimation algorithm which statistically correlates and decodes the outputs from both data structures to accurately infer the flow size distribution for any subpopulation.

The contributions of this work are three fold. First, we identify the value of the ability to specify the subpopulation of interest *a posteriori*, i.e. after data collection, when measuring SubFSD and other derived statistics. Although this has been an implicit requirement in many monitoring problems, we first explicitly target our design efforts at this facet of the problem, arriving at a very efficient and accurate solution. The second contribution is the architectural innovation of combining two existing FSD measurement techniques, one based on sampling and the other based on data streaming, for solving the new problem of SubFSD. The third and chief contribution of this work lies in the design of the estimation algorithm. To the best of our knowledge this is the first algorithm in the area of network measurement that uses observations from two or more data collection modules to perform a joint estimation.

The rest of this paper is organized as follows. The next section provides the background work in the area, including an overview of previous solutions that our techniques build upon. Section 3 presents the overall architecture of our solution and a description of the various modules. Section 4 presents our estimation algorithm, along with formal statements and proofs of various results derived from a probabilistic modeling of the data collection processes used in our algorithm. Experimental evaluations using publicly available traces are presented in Section 5. The experiments demonstrate that our algorithm achieves very high level of accuracy in estimating the SubFSD for various subpopulations. A survey of related work is presented in Section 6 and we conclude in Section 7.

2. BACKGROUND

Maintaining per-flow state at routers would trivially provide an exact solution (i.e., without any losses) to the SubFSD problem. Unfortunately, it is impractical to maintain per-flow state information on high speed links (OC-192 and above) since a flow-based data structure would be too large to be held in expensive SRAM and constrained by the slow memory access speeds of DRAM. Therefore, all solutions that estimate flow statistics at high speeds incur some information loss during data collection. While sampling appears as the most intuitive solution to deal with the extremely large number of packets, more sophisticated data-collection techniques such as *data streaming* have been proposed recently. Data streaming [7] 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 key idea here is to retain as much information *pertinent to the queries* as possible during data collection.In this section, we provide summaries of the techniques for inferring the FSD from sampled data or data streaming. A survey of other related work is presented in Section 6.

Sampling-based approaches. This body of work focuses on inferring the distribution from sampled flow statistics [1, 2, 8, 9]. One can also infer the subpopulation flow size distribution from the sampled traffic since it is easy to obtain the samples corresponding to the subpopulation of interest from the complete set of samples. Duffield et al. [1, 9] study the statistical properties of packet-level sampling on real-world Internet traffic and propose an algorithm to infer the flow distribution from sampled statistics. After showing that the naive scaling of the flow distribution estimated from the sampled traffic is generally not accurate, the authors propose an *Expectation Maximization* (EM) algorithm to iteratively compute a more accurate estimate.

The EM algorithm is a standard statistical tool, used for estimation in situations where the amount of data is insufficient for a straightforward computation. Since high speed monitoring applications have to inevitably deal with lossy data collection, the EM algorithm has repeatedly surfaced as the mechanism of choice for estimation in such scenarios. We provide a more formal description of the EM algorithm in Section 4.

Recent research has shown that using the sampled data to estimate the FSD is not very accurate, and these inaccuracies will continue to plague any extension of the approach to SubFSD. In particular, it has been shown that there are significant limitations fundamental to this approach due to the information loss in packet-level sampling [2]. Our experimental evaluation presented in Section 5 uses a sampling based solution as a baseline to demonstrate the improvements in accuracy achieved by our proposed solution.

Streaming-based approaches. In a more recent work by Kumar et al. [3], a data streaming based approach has been used to circumvent the limitations of sampling. It uses an array of counters for data collection, updated at each packet arrival. A hash of the flow label is used to pick the counter to be incremented, but there is no explicit mechanism to handle collisions, resulting in fast but "lossy" data collection. The estimation phase uses an iterative EM algorithm that inverts the process of collisions in hashing to estimate the most likely distribution of flow sizes that would result in the observed distribution of counter values. This solution provides high accuracy estimates, with typically an order of magnitude smaller errors than sampling based solutions.

Although the data streaming approach provides accurate estimates of the FSD for the complete population, it discards the flow labels to save space, thus restricting the scope of any post-processing estimation to only the complete population of flows in the monitored traffic. It cannot be extended to estimate the SubFSD for arbitrary subpopulations since one would need to know all subpopulations of interest in advance and run an instance of the data streaming based solution [3] for each such subpopulation. This is impractical because there are a large number of possible subpopulations of interest, ranging from different protocol types (HTTP, FTP, etc.) to different prefix ranges in source or destination IP address (corresponding to traffic originating from, or destined to, customers or peers of an ISP etc.). Also, quite often the subpopulation of interest is not known until after the act of data collection.

3. SOLUTION ARCHITECTURE

Our solution leverages the strengths of both sampling and data streaming. Since sampling retains the headers of sampled packets, it provides the flexibility of choosing arbitrary subpopulations for SubFSD estimation after the act of data collection. However, sampling is not an accurate source of information about the sizes of various flows, and for this we turn to the more efficient technique of data streaming. Our estimation algorithm then correlates the information from these two sources to produce accurate estimates of SubFSD. In this section, we describe the architecture of our solution and the design of its component modules.

3.1 Overall architecture

The overall architecture is shown in Figure 1. The data collection phase involves two modules operating in parallel – an online streaming module operating on *all* the packets, and a NetFlow-like sampling module that samples a small percentage of the traffic and retains information about only the sampled packets. The online streaming module is a simple array of counters, used previously to estimate the FSD of complete population [3]. As discussed earlier, this module does not retain any information about the packet headers, and cannot be used for estimating flow size distribution of subpopulations by itself. The sampling module is similar to NetFlow that maintains per-flow state for the sampled packets.

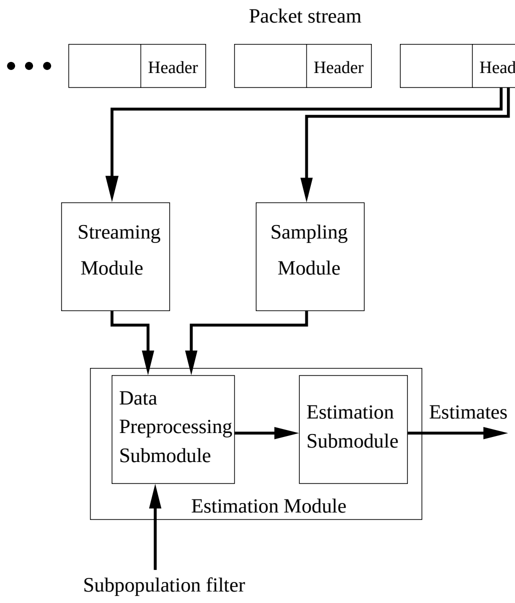


Figure 1: System model for using both data-streaming and sampling for data collection and estimation.

The measurement proceeds in epochs, with both data collection modules starting with a clean slate at the beginning of each epoch and paging the raw data to persistent storage at the end of each epoch. The data collected during an epoch is available for processing immediately at the end of the epoch. However, in practice, higher-level applications or users might decide to query this data at a much later time. Thus, the output of the data collection modules may need to be stored in the persistent storage, to be retrieved later for estimation.

The estimation module is composed of a preprocessing submodule that takes the data collected by the sampling and streaming modules, along with a specification of the subpopulation of interest, and produces a composite data structure. This preprocessed data is the input to our estimation submodule that produces the final estimate of the SubFSD for the specified subpopulation.

We emphasize the difference between the definition of flow label and the specification of a subpopulation of interest. The former refers to picking the fields of the IP header that will be stored for each sampled packet or hashed to pick an index in the counter array. The latter refers to a specific filter that defines a subpopulation. For example, the flow label can be defined as the four-tuple of source IP, destination IP, source port and destination port. A possible filter here would be *"source_port=80"*, thus specifying all webserver-to-client flows as the subpopulation of interest. The flow label needs to be defined *a priori* while the filters specifying the various subpopulations can be chosen at query time. The rest of this section describes each of these modules in detail.

3.2 Sampling module

The sampling based data-collection module in our system is the same as Cisco NetFlow, with one minor modification. To implement the semantics of epoch-based measurements, we require that all flow records be terminated at epoch boundaries. This extension has already been proposed by Estan et al. [10], and is essential for providing statistical guarantees of accuracy[1]. The sampling module maintains per-flow records for all sampled packets. At the end of each epoch, the flow label and flow size in packets from each record is summarized in a list and sent to the estimation module or to a persistent storage device.

3.3 Online streaming module

The online streaming module is the same as proposed first by Kumar et al. [3] – an array of counters, indexed by a hash function with range equal to the size of this array of counters. On each packet arrival, the flow label is hashed to generate an index into this array and the corresponding counter is incremented. There is no attempt to detect or avoid collisions. As a result, several flows might increment the same counter due to collisions in hashing. At the end of the measurement epoch, the entire array is paged out, to persistent storage or the estimation module.

The array of counters can be implemented using the efficient implementation of statistical counters proposed by Ramabhadran and Varghese [11]. 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, their value is added to the larger counter in DRAM, and the counter in SRAM is reset to zero. A more detailed discussion of the streaming module is available in the original paper that proposed this data structure [3].

3.4 Estimation module

The estimation module operates on the data collected by the sampling and the streaming modules to produce an estimate of the SubFSD. It is made up of two submodules — a data preprocessing submodule and the estimation submodule. The preprocessing submodule uses a filter specifying the subpopulation of interest to synthesize the data from the two modules into a composite data structure. Since the NetFlow records contain enough information to reconstruct the flow labels of all sampled flows, the hash function from the online streaming module can be used to associate each NetFlow record with a unique index in the array of counters. Also the *subpopulation filter* can be used to further classify

[1]Our solution does not require the other extensions proposed by Estan et al. [10], but it will work seamlessly if these extensions are implemented, and will also benefit from the advantages of improved sampling rates offered by these extensions.

Index (i)	Counter value (v_i)	list of sampled flow sizes from S and $\overline{S}$ (q_i)
1	1	-
2	20	$< 3, S >, < 1, \overline{S} >$
3	2	$< 1, S >$
4	1	$< 1, \overline{S} >$
5	0	-
6	5	$< 1, S >$
$\vdots$	$\vdots$	$\vdots$

Figure 2: Sample output of the preprocessing module.

the sampled flows as either belonging to the subpopulation S or its complement $\overline{S}$. The subpopulation filter can be a set of flow classification rules such that a flow matching any of the rules is defined to be in S and a flow matching none of these rules is in $\overline{S}$. Since the output of the sampling module contains only a small fraction of the total traffic, and is further aggregated into flows, this flow classification operation has negligible computational cost.

Figure 2 shows an example output of the preprocessing submodule. The first counter has a value 1 but has no associated sampled flows, because the only packet hashing to this index was not sampled. The second counter has a value of 20 and is associated with two sampled flows, one from S with 3 sampled packets and the other from $\overline{S}$ with just one sampled packet. The third counter has a value of 2 and is associated with a single flow from S with one sampled packet. Because sampling misses a large percentage of packets, the list of samples associated with any counter does not provide a complete description of all the flows hashing to that counter. However the sampled flow sizes do provide some information about the possible values of the actual flow sizes (before sampling), and more importantly, their membership in S or $\overline{S}$. The estimation submodule exploits this information to compute accurate estimates of FSD for S and $\overline{S}$. The details of this estimation algorithm and proofs of various results used in its operation are the subject of the next section.

4. THE ESTIMATION ALGORITHM

In this section, we describe our algorithm for estimating the subpopulation flow distribution from the observed data. We begin with a formal problem statement which also introduces definitions and notations used in the rest of this section. Section 4.2 formally describes the data collected by the two modules and the operations performed in the preprocessing submodule. Section 4.3 presents the estimation algorithm, derivations of results used in this algorithm, and a discussion of its computational complexity. Section 4.4 discusses the size of the array of counters.

4.1 Problem statement

The problem of computing the distribution of flow sizes for a given subpopulation S of the aggregate traffic can be formalized as follows. The set of possible flow sizes is the set of all positive integers between 1 to z, where z is the largest observed counter value – an upper bound on the maximum possible flow size in the monitored traffic. Among the flows belonging to S, 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 distribution of flow sizes in S is denoted by $\phi = \{\phi_1, \phi_2, \cdots, \phi_z\}$.

Let $\overline{S}$ be the complement of S, i.e., the set of all flows not in S. Although our goal is to estimate only the size distribution of flows in S, interestingly, we have to estimate it jointly with the size distribution of flows in $\overline{S}$, since these two distributions are statistically intertwined with each other. We denote the total number of flows in $\overline{S}$ as n', and the number of flows in $\overline{S}$ that have i packets as n'_i. The distribution of flow sizes in $\overline{S}$ is given by $\phi' = \{\phi'_1, \phi'_2, \cdots, \phi'_z\}$, where $\phi'_i = \frac{n'_i}{n'}$. Finally, we denote the joint distribution of flow sizes in S and $\overline{S}$ as Θ, where $\Theta = \{n, \phi, n', \phi'\}$. Our goal is to design an efficient algorithm to estimate Θ on a high-speed link (e.g., OC-192 or OC-768) with high accuracy, for any arbitrary S.

4.2 Observed data and preprocessing

Let m be the size of the counter array in the streaming module. Let v_i be the value of the counter at index i in this array. Let q_i be the set of sampled flows associated with the index i using the hash function from the streaming module. In other words, q_i is the set of flows in the sampled data whose flow labels hash to i. Recall that the preprocessing module produces the values v_i and q_i for each index i ($1 \leq i \leq m$) in the counter array. Since sampling may miss some flows completely, some of the q_i will be empty even if v_i, the value of the associated counter, is not zero.

The set of flows in q_i can be further classified as belonging either to the set S or to $\overline{S}$. The only information we use from each of the sampled flows in q_i is their sizes (number of sampled packets) and a Boolean variable indicating whether this flow is in S or not. Thus q_i can be reduced to a list of 2-tuples, such that the first element in each tuple represents the size of the sampled flow and the second element indicates whether the flow is in S or $\overline{S}$. For example, if q_i contains three (sampled) flows, among which two have size 1 and belong to S and the third has size 2 and belongs to $\overline{S}$, then $q_i = \{<1, S>, <1, S>, <2, \overline{S}>\}$.

To obtain Θ, we only need to know, for each index i, the sizes of individual flows that contribute to the corresponding counter value v_i, and the subpopulation that each of these individual flows belong to[2]. However, this information is lost in the data collection process. Collisions in hashing imply that we do not know the sizes of individual flows that contribute to the total value observed at any counter. During sampling, a substantial fraction of flows are missed completely. Even for the flows that do get sampled, the observed sizes in the sampled data are much smaller than the actual sizes of these flows in the original traffic. Therefore, the observed data is not sufficient to allow an exact calculation of Θ.

When we see a counter value v_i, there can be many different combinations of flows from S and $\overline{S}$ that add up to the value v_i. We refer to each such combination as a *split pattern*. Similar to the representation of q_i above, each split pattern can be represented as a multiset of 2-tuples. Each tuple corresponds to a flow in the actual traffic (before sampling), with the first element of the tuple representing the actual size of this flow (before sampling) and the second element of the tuple denoting whether or not the flow is in S. For example, if a counter of size 20 is split into four colliding flows, one each of size 10 and 2 from S and another two of size 3 and 5 from $\overline{S}$, the split pattern can be represented as $\{<10, S>, <2, S>, <5, \overline{S}>, <3, \overline{S}>\}$. We consider two split patterns as equal if their corresponding multisets of 2-tuples are equal to each other. Note that there are several possible split patterns for any observed counter value. Furthermore, for a given split pattern, there can be multiple ways of matching its constituent flows with the observed flows in the sampled data q_i. We return to this last point in our description of the estimation algorithm in the next section.

[2]A simple tabulation of this knowledge would result in an exact estimate of Θ.

4.3 Estimation algorithm

Our goal, which is common in most statistical estimation problems, is to find the maximum likelihood estimation (MLE) of Θ based on the observations. In other words, we would like to find $argmax_\Theta P(\Theta|\mathcal{Y})$, the Θ that maximizes the probability of seeing the observations $\mathcal{Y}$. However, we are not able to obtain this MLE directly for two reasons. First, there does not seem to be a closed-form formula for evaluating the likelihood function $P(\Theta|\mathcal{Y})$. Second, even if this probability can be somehow evaluated (say through an oracle), such an evaluation would have to be performed over a prohibitively large number of possibile values of Θ.

We adopt a standard methodology called Expectation Maximization (EM) to cope with this problem. EM allows for the approximation of MLE in situations where a direct computation of MLE is analytically intractable or computationally prohibitive, but, where the knowledge of some (missing) values for some auxiliary parameters can make the computation of MLE analytically tractable or computationally affordable. Intuitively, an EM algorithm proceeds as follows. It starts with a guess of the parameters we would like to estimate. Typically the expectations of the missing values can be computed conditioned on these parameters. Then we replace the occurrences of the missing values in the likelihood function $P(\Theta|\mathcal{Y})$ by their conditional mean. Finally, maximizing the likelihood function with these replacements will result in a new and better estimate of the parameters. This process will iterate multiple times until the estimated parameters converge to a set of values (typically a local MLE).

4.3.1 *Our EM algorithm*

A simplified version of our EM algorithm for estimating $\Theta = \{n, \phi, n', \phi'\}$ is shown in Figure 3. The algorithm begins with an initial guess of Θ (line 2 in Figure 3); a discussion about picking the initial guess is provided in Section 4.3.3. Each iteration of the algorithm proceeds as follows. For each of the m counters, we perform the following steps. Recall that v_i is the value of the counter at index i and q_i is the list of 2-tuples representing the sampled data associated with this counter. Let $\Omega_{(v_i,q_i)}$ represent the set of feasible split patterns for the counter at index i. Then for each split pattern $\alpha \in \Omega_{(v_i,q_i)}$, we compute $P(\alpha|\Theta, v_i, q_i)$, the probability of occurrence of α conditioned upon the current estimate of Θ and the observations v_i and q_i associated with this counter. The contribution of this event α, weighted by its probability, is credited to the flow counts of all constituent flows in α. Suppose α=$\{<a_1, S>, <a_2, S>, ..., <a_l, S>, <a'_1, \overline{S}>, <a'_2, \overline{S}>, ..., <a'_r, \overline{S}>\}$, i.e., the split pattern α corresponds to the event that l flows from S with sizes

```
Input: our observation $\mathcal{Y} = \{< v_i, q_i >\}_{1 \le i \le m}$
Output: (local) MLE for $\Theta$

1.  Initialization: pick initial distribution $\Theta^{ini}$
     as described in Section 4.3.3.
2.  $\Theta^{new} := \Theta^{ini}$;
3.  while (convergence condition is not satisfied)
4.    $\Theta^{old} := \Theta^{new}$;
5.    for $i$ :=1 to $m$
        /*$\Omega_{(v_i,q_i)}$ is the set of all split patterns
        consistent with the observation $< v_i, q_i >$*/
6.      foreach $\alpha \in \Omega_{(v_i,q_i)}$
          /*Suppose $\alpha = \{< a_1, S >, < a_2, S >, ..., < a_l, S >,$
                $< a'_1, \overline{S} >, < a'_2, \overline{S} >, ..., < a'_r, \overline{S} >\}$*/
7.        for $j$ := 1 to l
8.          $n_{a_j} := n_{a_j} + P(\alpha|\Theta, v_i, q_i)$
9.        for $j$ := 1 to r
10.         $(n_{a'_j})' := (n_{a'_j})' + P(\alpha|\Theta, v_i, q_i)$
            /* The formula for computing $P(\alpha|\Theta, v_i, q_i)$
            is derived in Section 4.3.2*/
11.       end
12.     end
13.   end
      /* Compute the new estimate of $n$ and $n'$. */
14.   $n^{new} := \sum_{i=1}^{z} n_i$
15.   $(n')^{new} := \sum_{i=1}^{z} (n'_i)$
      /* Normalize the flow counts $n_i$, $n'_i$ into $\phi$ and $\phi'$,
      and clear the counter values for the next iteration.*/
16.   for $i$:=1 to $z$
17.     $\phi_i^{new} := n_i / n^{new}$
18.     $(\phi'_i)^{new} := n'_i / (n')^{new}$
19.     $n_i := 0$
20.     $n'_i := 0$
21.   end
22. end
```

Figure 3: EM algorithm for computing the SubFSD

$a_1, a_2, ..., a_l$ and r flows from $\overline{S}$ with sizes $a'_1, a'_2, ..., a'_r$ collide at the counter in question. Then we increment each of the counts n_{a_j}, $j \in \{1, 2, ..., l\}$ and n'_{a_j}, $j \in \{1, 2, ..., r\}$ by $P(\alpha|\Theta, v_i, q_i)$. This tabulation process is illustrated through the following example.

Say the value of a counter at some index i is $v_i = 2$, and the sampled flow set is $q_i = \{< 1, S >\}$. Then $\Omega_{(v_i,q_i)}$, the set of split patterns that are consistent with the observation contains three elements, contains (i) $\{< 2, S >\}$, (ii) $\{< 1, S >, < 1, S >\}$, and (iii) $\{< 1, S >, < 1, \overline{S} >\}$. Note that other possible split patterns of 2, such as $< 2, \overline{S} >$, are excluded since they are not consistent with the observation $\{< 1, S >\}$. Suppose the respective probabilities of the events (i), (ii), and (iii), are 0.5, 0.3, and 0.2. A probability of 0.5 for event (i) implies that, with this probability, exactly one flow of size 2 from subpopulation S hashed to counter i, and exactly one packet from this flow was sampled. To tabulate this, we should increment our count of n_2, the number of flows in S that have size 2, by 0.5. Similarly, the probabilities of events (ii) and (iii) are used to increment the counts of the constituent flows. In all, we execute the following three sets of operations, corresponding to these three events.

- (i) $n_2 := n_2 + 0.5$;
- (ii) $n_1 := n_1 + 0.3$; $n_1 := n_1 + 0.3$
- (iii) $n_1 := n_1 + 0.2$; $n'_1 := n'_1 + 0.2$

Summarizing all 3 cases, we credit 0.5 to the counter n_2, 0.3 + 0.3 + 0.2 = 0.8 to the counter n_1, and 0.2 to the counter n'_1. Thus at the end of the loop for counter i (line 5), the algorithm has considered all split patterns consistent with the observations and used the respective probabilities of each of these splits to increment the counts of the constituent flow sizes. After processing each index $i \in 1, 2, ..., m$ in this manner, we obtain a final tabulation of the counters for the flow counts n_1, n_2, ..., n_z, n'_1, n'_2, ..., n'_z, and obtain n^{new}, $(n')^{new}$. We then renormalize the counts into new (and refined) flow distributions ϕ^{new}, $(\phi')^{new}$.

Note that if two indices i and j have the same associated observations, i.e., $v_i = v_j$ and $q_i = q_j$, then the entire set of operations for index i will be repeated for index j. An obvious optimization is to group all such indices together, and perform this set of operations for the group just once. Due to the "Zipfian" nature of internet traffic, there are a large number of small flows which cause similar observations at various indices. In other words, the number of unique configurations of (v_i, q_i) pairs observed in real traffic is quite small (approximately a few thousand). Thus grouping identical observations together reduces the length of the loop at line 5 in Figure 3 from m, which can be as large as a few millions, to a few thousand.

4.3.2 Computing the probability $P(\alpha|\Theta, v, q)$

Let i be an arbitrary index, and let $v = v_i$ and $q = q_i$ be the observations at this index. In the rest of this section, we drop the subscript i for better readability. We now derive the formula for computing $P(\alpha|\Theta, v, q)$, the probability of a particular split pattern α given the observation v, q and the *a priori* flow size distribution Θ, estimated in the preceding iteration. This computation is the most important step in our EM algorithm (lines 8 and 10 in Figure 3).

To be able to compute this probability, we need to introduce the concept of *matching* split patterns with the observed sampled flows q. Consider one possible split pattern α of the counter value v. A matching between α and q specifies exactly one of the possible ways in which the actual flows in α would result in the observation q after sampling. There can be multiple possible ways of matching α with q, as we will show shortly. We represent each matching as a list of 3-tuples. Each 3-tuple associates exactly one element from α with at most one element from q. Recall that elements in α are 2-tuples where the first element is the actual size of the flow and the second element indicates whether the flow belongs to S or $\overline{S}$. Similarly, elements in q are 2-tuples denoting the size of a sampled flow and its membership in S or $\overline{S}$. Now, in each 3-tuple in a matching between α and q, the first element represents the size of the flow before sampling and the third element, its size after sampling as observed in q. The middle element denotes whether the flow belongs to S or $\overline{S}$. If none of the packets from a specific flow in the split pattern α are sampled, the corresponding third element in the matching is 0. For example, suppose $\alpha = \{< 2, S >, < 10, S >, < 3, \overline{S} >, < 5, \overline{S} >\}$ and $q = \{< 3, S >, < 1, \overline{S} >\}$. The only two possible ways of matching α and q are:

- $\{< 2, S, 0 >, < 10, S, 3 >, < 3, \overline{S}, 1 >, < 5, \overline{S}, 0 >\}$
- $\{< 2, S, 0 >, < 10, S, 3 >, < 3, \overline{S}, 0 >, < 5, \overline{S}, 1 >\}$

The first matching is the event where the flow of size 10 from S has two packets in the sampled data set, the flow of size 2 from S has no sampled packets, and the two flows from $\overline{S}$ of sizes 3 and 5 contribute 1 and 0 packets respectively to the sampled data. The second matching differs from the first in that the sampled flow of size 1 from $\overline{S}$ is associated with the actual flow of size 5 instead of the other flow from $\overline{S}$ of size 3.

To distinguish between matchings, we need to assign them a *canonical form*. The canonical form of a matching π is specified as $< a_1, S, b_1 >, < a_2, S, b_2 >, ..., < a_l, S, b_l >, < a'_1, \overline{S}, b'_1 >, < a'_2, \overline{S}, b'_2 >, ..., < a'_r, \overline{S}, b'_r >$, where $a_1 \leq a_2 \leq ... \leq a_l$ and $a'_1 \leq a'_2 \leq ... \leq a'_r$, and $b_i \leq a_i, b'_j \leq a'_j$ for $1 \leq i \leq l, 1 \leq j \leq r$ (flow size after sampling should never be bigger than before sampling). In other words, we group flows from S and $\overline{S}$ together and in each group the flow sizes in the split pattern are sorted in increasing order. Two matchings π_1 and π_2 are considered the same if their canonical forms are identical. We say that a matching π is consistent with the observations v and q, if the a's add up to v and the non-zero b's have a one-to-one correspondence — in size and in their membership in S or $\overline{S}$ — with the observed sampled flows in q. We denote the set of all distinct consistent matchings of the split pattern α with the observation q as $\Pi(\alpha, q)$. With these definitions and notations, we are now ready to characterize the probability $P(\alpha|\Theta, v, q)$.

THEOREM 1. *Let w be the sum of all sampled flow sizes in q. The probability of occurrence of a split pattern α, given an observation $< v, q >$ and an estimate of Θ, is given by:*

$$P(\alpha|\Theta, v, q) = \frac{\sum_{\pi \in \Pi(\alpha,q)} P(\pi|\alpha, w) P(\alpha|\Theta, v)}{\sum_{\beta \in \Omega(v)} \sum_{\pi \in \Pi(\beta,q)} P(\pi|\beta, w) P(\beta|\Theta, v)}$$

where $P(\alpha|\Theta, v)$ is given by Lemma 1 and $P(\pi|\alpha, w)$ is given by Lemma 2.

PROOF. $P(\alpha|\Theta, v, q) = P(\alpha|\Theta, v, q, w)$

$$= \frac{P(q|\alpha, \Theta, v, w) P(\alpha|\Theta, v, w)}{\sum_{\beta \in \Omega(v)} P(q|\beta, \Theta, v, w) P(\beta|\Theta, v, w)}$$

$$= \frac{P(q|\alpha, w) P(\alpha|\Theta, v)}{\sum_{\beta \in \Omega(v)} P(q|\beta, w) P(\beta|\Theta, v)}$$

$$= \frac{\sum_{\pi \in \Pi(\alpha,q)} P(\pi|\alpha, w) P(\alpha|\Theta, v)}{\sum_{\beta \in \Omega(v)} \sum_{\pi \in \Pi(\beta,q)} P(\pi|\beta, w) P(\beta|\Theta, v)}$$

The first equality holds since w is a function of q. The second equality is by Bayes' rule, with all terms conditioned on Θ, v, w. In the third equality, $P(q|\alpha, \Theta, v, w) = P(q|\alpha, w)$ because v is a function of α, and Θ does not matter once α is determined. Also, in the third equality, $P(\alpha|\Theta, v, w) = P(\alpha|\Theta, v)$ because once v is determined, w, the number of packets sampled from v, does not affect the probability of α. Finally, the last equality holds since $P(q|\alpha, w)$ is equal to $\sum_{\pi \in \Pi(\alpha,q)} P(\pi|\alpha, w)$ by the definition of $\Pi(\alpha, q)$. □

LEMMA 1. *Let $\Omega(v)$ be the set of all split patterns that add up to v. Then $P(\alpha|\Theta, v) = \frac{P(\alpha|\Theta)}{\sum_{\beta \in \Omega(v)} P(\beta|\Theta)}$, where $P(\alpha|\Theta)$ and $P(\beta|\Theta)$ can be computed using Lemma 3.*

The proof is a straightforward application of Bayes' rule, and is analogous to the proof of a similar theorem by Kumar et al. [3]. We omit it here.

LEMMA 2. *Let α be a split pattern $< a_1, S >, < a_2, S >, ..., < a_l, S >, < a'_1, \overline{S} >, < a'_2, \overline{S} >, ..., < a'_r, \overline{S} >$ that is consistent with observed counter value v (i.e., $\sum_{i=1}^{l} a_i + \sum_{j=1}^{r} a'_j = v$). Let $\pi =< a_1, S, b_1 >, < a_2, S, b_2 >, ..., < a_l, S, b_l >, < a'_1, \overline{S}, b'_1 >, < a'_2, \overline{S}, b'_2 >, ..., < a'_r, \overline{S}, b'_r >$ (in the canonical form) be a matching that is consistent with the split pattern α and the observation q. Then in a packet sampling process with any uniform sampling probability, the probability that the matching π happens given that α is the underlying split pattern is*

$$P(\pi|\alpha, w) = \frac{\binom{a_1}{b_1}\binom{a_2}{b_2} \cdots \binom{a_l}{b_l}\binom{a'_1}{b'_1}\binom{a'_2}{b'_2} \cdots \binom{a'_r}{b'_r}}{\binom{v}{w}} \quad (1)$$

PROOF. Let p be the sampling probability used in the packet sampling process. We have $P(\pi|\alpha, w) = P(\pi, w|\alpha)/P(w|\alpha) = P(\pi|\alpha)/P(w|\alpha)$. The last equality holds since the matching π determines the value of w (note that $w = \sum_{i=1}^{l} b_i + \sum_{j=1}^{r} b'_j$ for π to be consistent with the observation q). We know that $P(w|\alpha) = \binom{v}{w} p^w (1-p)^{v-w}$ since this is the probability that among a total of v packets, w are actually sampled with a sampling probability p. It remains to derive the probability $P(\pi|\alpha)$. Let e_i be the event that b_i packets are chosen from a flow of size a_i from S in the split pattern α, $i \in \{1, 2, ..., l\}$. Similarly, let e'_j be the event that b'_j packets are chosen from a flow of size a'_j from $\overline{S}$ in α, $j \in \{1, 2, ..., r\}$. Then, $P(e_i) = \binom{a_i}{b_i} p^{b_i} (1-p)^{a_i - b_i}$, and $P(e'_j) = \binom{a'_j}{b'_j} p^{b'_j} (1-p)^{a'_j - b'_j}$. Now note that π conditioned upon α is exactly $\left[\wedge_{i=1}^{l} e_i\right] \wedge \left[\wedge_{j=1}^{r} e'_j\right]$. Thus, we have $P(\pi|\alpha) = P\left(\left[\wedge_{i=1}^{l} e_i\right] \wedge \left[\wedge_{j=1}^{r} e'_j\right]\right) = \left[\prod_{i=1}^{l} P(e_i)\right]\left[\prod_{j=1}^{r} P(e'_j)\right] = \left[\prod_{i=1}^{l} \binom{a_i}{b_i} p^{b_i} (1-p)^{a_i - b_i}\right]\left[\prod_{j=1}^{r} \binom{a'_j}{b'_j} p^{b'_j} (1-p)^{a'_j - b'_j}\right]$. The second equality is due to the fact that the events e_1, e_2, ..., e_l, e'_1, e'_2, ..., e'_r are independent. The final term can be simplified as $\left[\prod_{i=1}^{l} \binom{a_i}{b_i}\right]\left[\prod_{j=1}^{r} \binom{a'_j}{b'_j}\right] p^w (1-p)^{v-w}$. Therefore,

$P(\pi|\alpha, w) = P(\pi|\alpha)/P(w|\alpha) = \frac{(\prod_{i=1}^{l} \binom{a_i}{b_i})(\prod_{j=1}^{r} \binom{a'_j}{b'_j})}{\binom{v}{w}}$. □

LEMMA 3. *Let α be a split pattern in which there are f_1 flows of size s_1, f_2 flows of size s_2, ..., f_k flows of size t_k from S, and f'_1 flows of size s'_1, f'_2 flows of size s'_2, ..., f'_k flows of size s'_q from $\overline{S}$. Given Θ, the a priori (i.e., before observing the values v, q) probability of occurrence of α is:*

$$P(\alpha|\Theta) = \left(e^{-n/m} \prod_{i=1}^{k} \frac{(n_{q_i}/m)^{f_i}}{f_i!}\right)\left(e^{-n'/m} \prod_{i=1}^{k} \frac{(n'_{q'_i}/m)^{f'_i}}{f'_i!}\right)$$

.

PROOF. Let $\mathcal{E}$ 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 from the set S collide at a counter, and let $\mathcal{E}'$ 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 from the set $\overline{S}$ collide at a counter. Kumar et al. [3] prove that $P(\mathcal{E}) = e^{-n/m} \prod_{i=1}^{k} \frac{(n_{q_i}/m)^{f_i}}{f_i!}$ and $P(\mathcal{E}') = e^{-n'/m} \prod_{i=1}^{k} \frac{(n'_{q'_i}/m)^{f'_i}}{f'_i!}$. Note that the two events $\mathcal{E}$ and $\mathcal{E}'$ are independent of each other. Therefore, the result follows from $P(\alpha|\Theta) = P(\mathcal{E} \wedge \mathcal{E}'|\Theta)$.

□

4.3.3 Obtaining an initial guess for Θ

Each iteration of our estimation algorithm uses the estimate Θ^{old}, produced by the preceding iteration, to compute a refined estimate Θ^{new}. The first iteration begins with an initial guess or seed Θ^{ini}. In our algorithm, we generate Θ^{ini} by using the observed distribution of counter values in the array of counters as the initial guess for both ϕ and ϕ'. The initial guesses for n and n' are generated by taking the total number of non-zero counters and partitioning this number in the ratio of the number sampled flows from S and $\overline{S}$ in the sampled data, respectively. We have tried more sophisticated heuristics to generate Θ^{ini}. However, since our algorithm converges very fast and smoothly even with this simple Θ^{ini}, shown in Section 5.3, the benefits of having a better Θ^{ini} seems marginal.

4.3.4 The Expectation and Maximization steps

We now explain why our algorithm is an instance of the EM algorithm, and hence guaranteed [12] to converge to a set of local MLEs. Let $\gamma_{i,j,k}$ denote the number of flows from S that have an actual size i, but hash to counters of size j contributing k sampled packets. Similarly, we can define $\gamma'_{i,j,k}$ for flows from S'. These two sets of variables, $\gamma = \{\gamma_{i,j,k}\}$ and $\gamma' = \{\gamma'_{i,j,k}\}$, are the missing data whose knowledge would allow us to compute the precise value of Θ. In the expectation step, our algorithm uses a previous estimate of Θ (i.e., Θ^{old}) to compute the *expected* value of γ and γ'. In the maximization step, the new value of Θ (i.e., Θ^{new}) is computed as: $n = \sum_{i,j,k} \gamma_{i,j,k}$, $\phi_i = \sum_{jk} \gamma_{i,j,k}/n$, $n' = \sum_{i,j,k} \gamma'_{i,j,k}$, and $\phi'_i = \sum_{jk} \gamma'_{i,j,k}/n'$. It can be shown that this choice of Θ^{new} indeed maximizes the data likelihood function $p(\gamma, \gamma', \Theta|\mathcal{Y})$, where $\mathcal{Y}$ is the observed data.

4.3.5 Computational complexity of the algorithm

The computation of $P(\alpha|\Theta, q, v)$ in our algorithm (lines 8 and 10 in Figure 3) using theorem 1 involves enumerating all split patterns that give rise to the observed counter value v and then for each such split pattern, enumerating all possible matchings of the split pattern with the observed sampled data q. The number of total cases is immense for large values of v. The "Zipfian" nature of flow-size distribution comes to our rescue here. Since most flows have just a few packets, the probability of three or more flows, with more than 10 packets each, colliding at the same counter is negligible. Thus we can safely ignore cases with more than two sampled flows associated with a counter. Furthermore, to reduce the complexity of enumerating all events that could give rise to a large counter value (say larger than 300), we enumerate only the cases involving the collision of up to 3 unsampled flows from each of S and $\overline{S}$ at such indices. Thus the asymptotic computational complexity is $O(v^3)$ for counters with value greater than 300. With similar justifications we enumerate up to 4 collisions each from S and $\overline{S}$, for counters with value smaller than 300. 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 corresponding to the largest sampled flow associated with such counters. 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.

Trace	# of flows	# of packets
USC	986,702	15,730,938
UNC	1,133,150	27,507,496
NLANR	55,515	158,243

Table 1: Traces used in our evaluation.

4.4 The size of the array of counters

The number of flows in high speed links is known to be highly variable, especially in adversarial scenarios such as a DDoS attack. The number of flows may increase by more than an order of magnitude in such situations. Provisioning our sampling and streaming modules for this worst case would clearly be very expensive. Yet a system naively provisioned for just the average case will not be able to keep up with the traffic in the worst case. For example, current implementations of NetFlow often run out of memory for keeping flow records and start dropping them. On the sampling slide, this problem has been identified and addressed in a recent work by Estan et al. [10]. In this work, the authors propose a comprehensive revamp of NetFlow that uses an adaptive sampling scheme to keep up with high flow counts. On the data streaming slide, multiresolution schemes have been proposed to deal with the same problem [3, 4]. In particular, the work by Kumar et al. [3] addresses this issue in the context of estimating FSD in detail. The authors show that their counter array based scheme is very accurate when the number of counters is roughly the same ($\pm 50\%$) as the number of flows. They also propose a multiresolution extension to their scheme to deal with the high variability in the number of flows. The algorithm proposed in this paper can be extended in a similar fashion to solve the same problem. Since this issue has been adequately addressed in the literature, in the rest of this paper, we will use a counter array size that is the closest power of 2 to the number of flows in the corresponding measurement epoch, as suggested in the work of Kumar et. al. [3].

5. 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 from sampled data [1]. Since our algorithm uses the data collected in the streaming module in addition to the sampling based data, its estimates have higher accuracy. For this comparison, we implemented the sampling based estimation mechanism proposed by Duffield et al. [1] and carefully verified that our implementation is faithful to the published description of the scheme.

We begin this section with a description of the various packet-header traces used in our evaluation, followed by a description of the evaluation metrics and visualization techniques used in the rest of the section. We then discuss the termination of the estimation algorithms and finally present experiments evaluating the accuracy of the estimates of Sub-FSD, the impact of sampling rates, and the estimates of derived statistics produced by the proposed algorithm.

5.1 Traffic traces

We use packet header traces of Internet traffic from three sources: University of North Carolina (UNC) (courtesy of

Prof. Jeffay), University of Southern California (USC) (courtesy of Prof. Papadopoulos), and NLANR [13]. The trace form UNC was collected at the 1 Gbps access link connecting the campus to the rest of the Internet, on Thursday, April 24, 2003 at 11:00 am. The second trace from USC was collected at their Los Nettos tracing facility on Feb. 2, 2004. The packet header traces from USC and UNC correspond to a few hours' worth of traffic. Since our mechanism is designed to operate in epochs on the order of a few hundred seconds, in our experiments, we used 500 second long segments taken from these two traces.

The NLANR trace was collected at an OC-3 link at the University of Auckland on December 8, 1999 for a duration of around two hours and nineteen minutes. All the above traces are either publicly available or available for research purposes upon request from their respective sources. Table 1 lists the number of flows and packets in each trace segment that was used in our experiments.

Following the discussion in Section 4.4, we use a counter array of size 64k (2^{16}) for the trace NLANR and an array of 1M (2^{20}) counters for the traces UNC and USC. For each trace, we perform the streaming and sampling operations just once to obtain the observations. Estimations for various subpopulations in a trace will all come from this observation. Throughout this section, we use a sampling rate of 10%, except for section 5.5 where we study the impact of sampling rate on estimation accuracy.

5.2 Evaluation metrics and visualization

For comparing the estimated flow size distribution with the actual distribution, we use the metric of *Weighted Mean Relative Difference (WMRD)*, defined as:

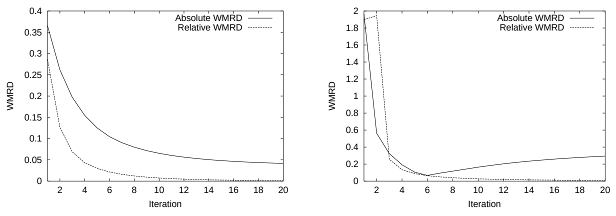
$$WMRD = \frac{\sum_i \frac{|n_i - \hat{n_i}|}{\left(\frac{n_i+\hat{n_i}}{2}\right)} \times \frac{n_i+\hat{n_i}}{2}}{\sum_i \left(\frac{n_i+\hat{n_i}}{2}\right)} = \frac{\sum_i |n_i - \hat{n_i}|}{\sum_i \left(\frac{n_i+\hat{n_i}}{2}\right)}$$

where n_i is the number of flows of size i and $\hat{n_i}$ is its estimate. WMRD assigns a weight of $\frac{n_i+\hat{n_i}}{2}$ to the relative error in estimating the number of flows of size i. Thus, this metric reflects the errors in estimating the number of large and small flows in proportion to their actual population. This metric was proposed and used by Duffield et al. [1], and subsequently used by Kumar et al. [3], for the same purpose of evaluating the accuracy of estimated flow distribution.

Since there are very few large flows, the tail of the flow distribution plot is very noisy. To obtain a more intuitive visual depiction of the flow distribution, we borrow a bucket-based smoothing technique from the work of Kumar et al. [3]. Instead of plotting the number of flows for every possible integer size, we plot the number of flows in an interval or bucket, normalized by the bucket size. 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.

5.3 Termination of the EM algorithms

The metric WMRD is also used to determine the terminating point of the estimation algorithm. Starting with the initial guess $\Theta^0 = \Theta^{ini}$, the WMRD between the estimate



(a) Proposed algorithm. (b) Sampling based estimation.

Figure 4: Absolute and relative WMRD.

Θ^i generated at the end of iteration i and the estimate Θ^{i-1} from the preceding iteration is computed. The algorithm terminates when this *relative* WMRD is below some predetermined threshold ϵ. For the evaluation of estimation accuracy and convergence speed of the EM algorithm, we also use the metric of *absolute* WMRD, defined as the WMRD between the estimate at the end of iteration i and the exact SubFSD. During actual operation, the absolute WMRD will not be known, and the termination condition has to rely solely on the relative WMRD.

Figure 4(a) shows the relative and absolute WMRD for various iterations of our EM algorithm. For both our EM algorithm and the sampling based estimation [1], the relative WMRD shows a monotonic decrease with more iterations. The absolute WMRD for the proposed mechanism also decreases monotonically with more iterations. Therefore, reducing the threshold ϵ will only reduce the absolute errors of the estimates, at the cost of an increased number of iterations. On the other hand, the absolute WMRD for sampling based estimation does not show a monotonic decrease with more iterations. Instead, as shown in Figure 4(b), the actual WMRD first decreases, reaching a local minimum after 6 iterations, and then gradually increases with further iterations. Furthermore, for different instances of the problem, this local minimum occurs at different iterations and for different values of relative WMRD[3]. A clairvoyant execution of the algorithm would terminate at just the right number of iterations where the absolute WMRD reaches its minimum. However, in practice, it is hard to reliably determine the optimal number of iterations. From our observation of various executions of the algorithm, we pick a value of $\epsilon = 0.07$ for the sampling based estimation algorithm, noting that smaller values of ϵ would result in larger absolute WMRD. For our proposed algorithm, we pick a more conservative value of $\epsilon = 0.002$. These termination conditions provide a fair and consistent comparison of the two mechanisms and are used in the rest of this section.

5.4 Accuracy of SubFSD Estimation

In this section, we report estimates for a number of subpopulations, specified by IP prefixes or port numbers. In

[3]We verified this aspect of the sampling based estimation algorithm with its authors [14]. The authors mention the possibility of an alternate termination rule that inspects the estimates after each iteration for the onset of oscillatory artifacts. Future research on this issue might produce better termination conditions.

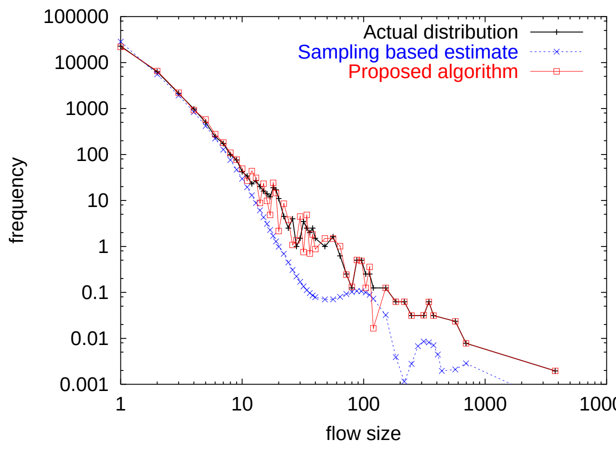


Figure 5: Actual, and estimated FSD for flows with source port 80 in the trace NLANR.

our first experiment, we specify all flows with source port 80 (i.e., traffic from web servers), in the trace NLANR, to be the subpopulation of interest. Figure 5 shows the actual FSD of this subpopulation and two estimates, the first generated by the sampling based estimation algorithm and the second generated using our mechanism. The tails have been smoothed for better visualization of the results as described above. Notice that the sampling based estimate is a "smooth" curve with significant deviations from the actual FSD while the estimates generated using our mechanism overlap almost completely with the actual FSD, accurately tracking the local variations in the actual FSD. Note that in this and all following figures, both the axes are on the logarithmic scale. Similar results were obtained over other traces using a variety of different specifications of the subpopulation of interest.

Figure 6 shows the results of subpopulation FSD estimation using trace UNC. The three subpopulations of interest here are DNS, HTTP and HTTPS flows, specified by their use of source or destination ports 53, 80 and 443 respectively. Interestingly, the DNS traffic constitutes less that 3% of the total flows, but accounts for most of the flows of size 1. On the other hand, the HTTP and HTTPS subpopulations have the largest number of flows of size 5 and 7 respectively. In all three cases, the curve for the sampling based estimates looks like a "smooth" average of the actual distribution. This behavior is clearly discernible in the figures for HTTP and HTTPS subpopulations, where sampling based estimation completely misses the "peaks" in the distributions. The proposed algorithm, on the other hand, manages to identify these peaks quite accurately. This ability can be important for applications such as worm detection, where the peak may reveal a fixed-size worm. Figure 7 shows similar behavior for the USC trace.

Intuitively, the reason for this behavior is as follows: with a sampling rate of 1/10, most flows of small sizes do not get sampled at all, or when sampled, exhibit a high variability in the number of sampled packets. Thus sampling based estimation tends to be inaccurate for small flow sizes and at best manages to estimate a smoothed average of the actual distribution. The situation is further aggravated when there are local maxima in the actual distribution. For example, in Figure 6(b), the large number of HTTP flows of size 5 may contribute anywhere between 0 and 5 packets to the sampled data. An estimation based on only the sampled data is quite likely to converge to an average case where some of these samples are "credited" towards the estimates of smaller flow sizes. Our algorithm, on the other hand, uses the counter value from the streaming data structure in addition to this sampled data. The counter array is quite accurate in gathering flow size information, and these counter values provide an accurate "anchor" for the estimation algorithm. In other words, our algorithm requires flow sizes in split patterns for any index to add up to the counter value at the index, hence using more detailed and fine grained information than pure sampling based estimation. Therefore, while our algorithm also exhibits some inaccuracies in estimates for small flow sizes due to sampling, which is discussed in detail below, it is significantly more accurate than sampling based estimation.

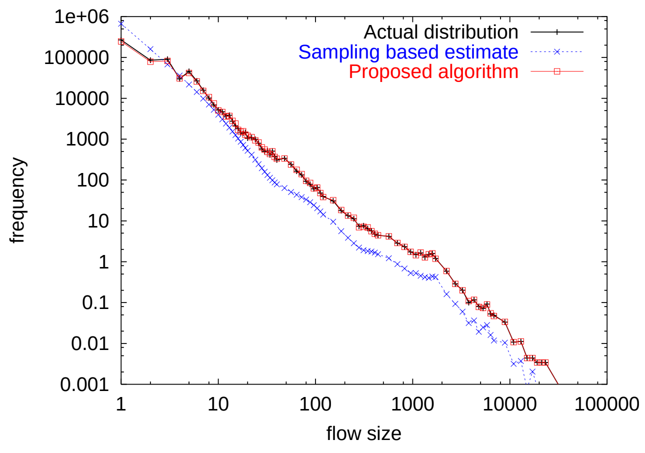
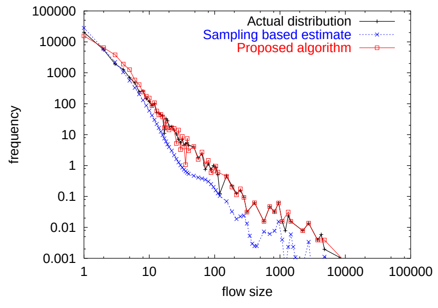


Figure 8: Actual, and estimated distributions for the subpopulation of flows from source IP matching the prefix 1.0.*.* in trace USC.

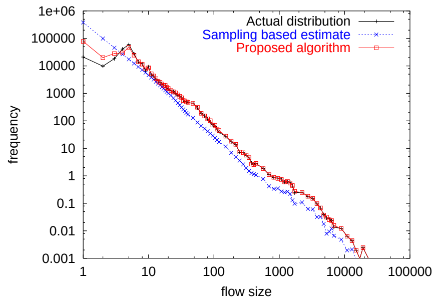
Figure 8 shows the FSD for the subpopulation of flows with source IP matching the prefix 1.0.*.* in the trace USC. Since the IP addresses in this trace were anonymized using a prefix preserving anonymization scheme, grouping traffic by prefix in the anonymized trace makes sense. This subpopulation accounts for 50 to 60 percent of the flows in the trace, and probably represents the outgoing traffic from machines in the same class-B address block.

5.5 Impact of sampling rate

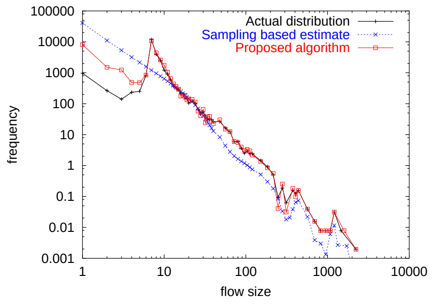
The amount of information lost in sampling increases when the sampling rate is reduced, leading to higher inaccuracies for the proposed algorithm as well as estimation based purely on sampled data. Table 2 shows the WMRD of estimates for various subpopulations in the trace NLANR. The sampled data was collected at sampling rates of 10%, 5% 2% and 1% in different runs of the experiment. While the accuracy of both mechanisms degrades with decreasing sampling rates, estimates generated from only the sampled data are 50 to 250% larger than those generated by the proposed algorithm. We observed similar improvements in accuracy across the other two traces over a variety of subpopulations.



(a) Estimate of FSD of all DNS flows.

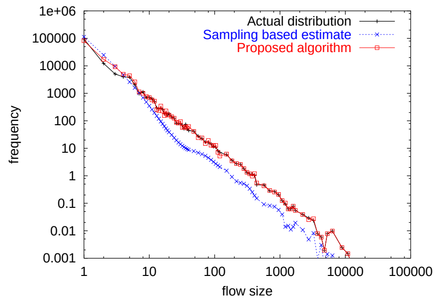


(b) Estimate of FSD of all HTTP flows.

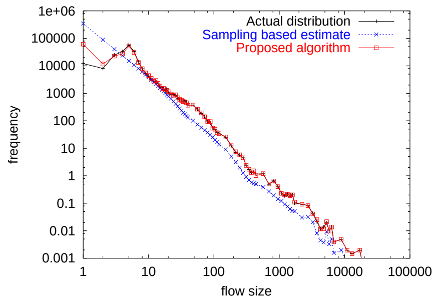


(c) Estimate of FSD of all HTTPS flows.

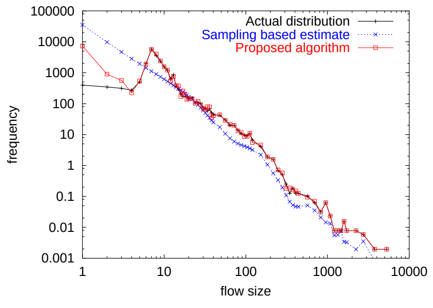
Figure 6: Actual, and estimated distributions for various subpopulations in the trace UNC.



(a) Estimate of FSD of all DNS flows.



(b) Estimate of FSD of all HTTP flows.



(c) Estimate of FSD of all HTTPS flows.

Figure 7: Actual, and estimated distributions for various subpopulations in the trace USC.

Trace	Sub-population	Actual value	Sampling based estimates	Proposed Algorithm
UNC	DNS	31932	38997.87	31389.68
	HTTP	277583	641874.11	330064.71
	HTTPS	27337	72174.95	37928.81
	Prefix 1.0	388326	673519.43	377266.40
USC	DNS	132396.00	161440.75	130686.73
	HTTP	244773.00	574610.23	283664.58
	HTTPS	24731.00	64523.73	32660.56
	Prefix 1.0	637949.00	1017053.20	599140.25

Table 3: Estimates of total number of flows for various subpopulations in traces UNC and USC.

5.6 Estimation of derived statistics

The knowledge of the SubFSD can be used to derive estimates for a number of other statistics for the subpopulation in question, such as the total number of packets, the total number of flows, the number of flows of size 1, etc. Here we show our results for the total number of flows. Table 3 lists the actual value of the total number of flows for various subpopulations in traces UNC and USC and their estimates produced by sampling based estimation and the proposed algorithm. There is an order of magnitude of improvement in estimates derived from our algorithm over estimates generated from the sampling based scheme. We observed similar improvements in accuracy for other derived statistics such as the total number of packets and the number of flows of size 1. The accuracy of our results demonstrates that the benefits of accurate estimation of SubFSD carry over to the estimation of other derived statistics on a subpopulation.

We emphasize that the comparisons in this section are not to point out the shortcomings of estimation algorithms that use only sampling based data [1]. The gain of accuracy in our mechanism comes from the additional use of the array of counters, which saves us from having to skip a large percent of traffic as done in the sampling-based approach which suffers a large information loss in sampling.

6. RELATED WORK

The streaming data structure of an array of counters has been used in a number of systems and applications within the area of network measurement and monitoring. For example, it has been the building block for detecting traffic changes [15], identifying large flows [16], and constructing Bloom filters [17] that allow both insertions and deletions (called counting Bloom filter) [18].

Previous work on estimating the FSD from sampled data [1] or using data-streaming techniques [3] was introduced in de-

	WMRD of Sampling based estimates				WMRD of proposed algorithm			
Sampling Rate	10%	5%	2%	1%	10%	5%	2%	1%
Flows from src. port 80	0.06726	0.18090	0.37872	0.46051	0.04494	0.07814	0.13766	0.14573
Flows to dst. port 80	0.08308	0.13228	0.30325	0.47049	0.03559	0.08422	0.02964	0.13430
Flows with src. or dst. port 80	0.06840	0.16869	0.35907	0.46274	0.04672	0.05114	0.11699	0.15240

Table 2: WMRD of estimates for some subpopulations in trace NLANR, with various sampling rates.

tail in Section 2. Both these solutions use EM for estimation. EM is also used in this paper, but in a very different way. While previously EM was used to either compensate for the information loss due to sampling or to compensate for the information loss due to hash collisions, here we have used EM to compensate for both kinds of information loss in a joint estimation procedure. Consequently, our algorithm is much more accurate than sampling based estimates, yet it retains the versatility of sampling based estimation because it makes full use of the packet header information retained in sampling.

Hohn and Veitch [2] discuss the inaccuracy of estimating flow distribution from sampled traffic, when the sampling is performed at the packet level, and show that sampling at the flow level leads to more accurate estimations. The study in [2] is focused mostly on the theoretical aspects of sampling.

Estan et al. [4] design a mechanism to count the total number of flows. The streaming data structure used in their solution is even more simple – an array of bits. A hash function is computed over the flow label of each arriving packet to generate an index into the bit array and the bit at the corresponding location is set to 1. The estimation phase models the insertion process as an instance of the Coupon Collector's problem [19] to derive a simple estimator for the total number of flows as a function of the size of the bit array and the number of bits set to 1. This solution also discards packet headers after updating the streaming data structure and hence cannot be used for estimating the total number of flows in arbitrary subpopulations.

7. CONCLUSIONS

As the interest in monitoring traffic at various points in the network increases, increasing amounts of research effort is being devoted to the design of mechanisms for data collection and analysis at high network speeds. A number of flow measurement problems such as estimating the FSD have been solved with varying degrees of success. Our work advances the state of art in the area. We identify the fundamental nature of the SubFSD problem, where the subpopulation of interest is specified a posteriori, and present an innovative architecture where two online data collection modules are used in tandem which is a non-trivial advance in traffic estimation techniques that have traditionally relied on making inferences from a single information source. Finally, the Bayesian procedure we develop to combine streaming information from multiple sources to obtain estimates, demonstrates the power of performing such joint estimation, and also provides insights into the relative merits of various information collection techniques.

8. 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] A. Kumar, M. Sung, J. Xu, and J. Wang, "Data streaming algorithms for efficient and accurate estimation of flow size distribution," in *Proc. ACM SIGMETRICS*, June 2004.

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

[5] P. Indyk, "Stable distributions, pseudorandom generators, embeddings and data stream computation," in *Proc. FOCS*, 2000, pp. 189–197.

[6] "Cisco netflow," http://www.cisco.com/warp/public/732/Tech/netflow.

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

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

[9] 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.

[10] C. Estan, K. Keyes, D. Moore, and G. Varghese, "Building a better netflow," in *Proc. ACM SIGCOMM*, Aug. 2004.

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

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

[13] "National Laboratory for Applied Network Research," Traces available at http://moat.nlanr.net/Traces/.

[14] N. Duffield, "Private communication," Oct. 2004.

[15] 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.

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

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

[18] 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.

[19] R. Motwani and P. Raghavan, *Randomized Algorithms*, chapter 3.6, Cambridge University Press, 1995.