Sketch Guided Sampling — Using On-Line Estimates of Flow Size for Adaptive Data Collection

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

***Abstract*— Monitoring the traffic in high-speed networks is a data intensive problem. Uniform packet sampling is the most popular technique for reducing the amount of data the network monitoring hardware/software has to process. However, uniform sampling captures far less information than can be potentially obtained with the same overall sampling rate. This is because uniform sampling (unnecessarily) draws the vast majority of samples from large flows, and very few from small and medium flows. This information loss on small and medium flows significantly affects the accuracy of the estimation of various network statistics.**

In this work, we develop a new packet sampling methodology called "sketch-guided sampling" (SGS), which offers better statistics than obtainable from uniform sampling, given the same number of raw samples gathered. Its main idea is to make the probability with which an incoming packet is sampled a decreasing sampling function f of the size of the flow the packet belongs to. This way our scheme is able to significantly increase the packet sampling rate of the small and medium flows at slight expense of the large flows, resulting in much more accurate estimations of various network statistics. However, the exact sizes of all flows are available only if we keep per-flow information for every flow, which is prohibitively expensive for high-speed links. Our SGS scheme solves this problem by using a small (lossy) synopsis data structure called counting sketch to encode the approximate sizes of all flows. Our evaluation on real-world Internet traffic traces shows that our sampling theory based the approximate flow size estimates from the counting sketch works almost as well as if we know the exact sizes of the flows.

I. Introduction

Accurate measurement and monitoring of data traffic traversing network links or routers is critical for network management and operation. For example, per-flow traffic measurement [1] is critical for network usage accounting, resource planning, traffic engineering, pricing and billing, and anomaly detection; flow size distribution [2] information can help detect an event (e.g., link failure) that causes the transition of the global network dynamics (e.g., traffic re-routing). With the rapid growth of the Internet, network link speeds have become faster every year to accommodate more Internet users. Accurate monitoring of the traffic on such high speed links is a challenging problem. For example, in a 40 Gbps (OC-768) link, the line card has a total of $25ns$ to forward a packet[1], and per-packet monitoring functions have to finish well within this time frame. Traditional implementations of such monitoring functions involve a search and update to a per-flow state (typically organized as a hash table), and this state has to be stored in fast (yet expensive) SRAM to keep up with the link speed. However, since this state can be very large (say hundreds of megabytes) at high link speeds, storing it in SRAM can be prohibitively expensive.

Packet sampling is the *de facto* technology for reducing the amount of data that has to be processed when monitoring high speed links and routers. Since its initial use at the monitoring infrastructure of NSFNET's T1 backbone in September 1991 [3], packet sampling has found increasing acceptance in network monitoring applications, and is currently supported by equipment vendors (e.g., Cisco's NetFlow [4]). By far its simplest form is *uniform packet sampling*, in which each packet is sampled independently with a fixed probability p, and only sampled packets trigger an update to the flow records. Deployed products, such as Cisco's NetFlow actually use a slight variant, namely periodic packet sampling (sampling every n_{th} packet), for ease of implementation. An advantage of uniform or periodic packet sampling is that it guarantees to reduce the traffic by a factor of $1/p$ in both long and short time scales, which is very important when the router CPU that processes sampled traffic (e.g., the hash table of flow records in NetFlow) operates under hard resource constraints[2].

[1]We assume a conservative packet size of 1000 bits here.

[2]Flow sampling, to be discussed in Sec. II, is a more attractive alternative to packet sampling for certain applications, but it does not guarantee the traffic reduction ratio in the same way assured by packet sampling.

While uniform (or periodic) sampling is very straightforward to implement, we discover that it captures far less information than can be potentially obtained with the same overall sampling rate p. This is because uniform sampling devotes resources in proportion to the size of a flow. However, the information gained from sample collection in this manner does not grow in proportion to the flow size. For example, sampling 1,000 packets from a flow of size 10,000 clearly contains far less information than sampling one packet each from 1,000 flows of size 1, while consuming the same amount of sampling resource (processing 1,000 sampled packets). This problem is exacerbated in Internet monitoring due to the Zipfian nature of the Internet traffic – there are very few large flows but they contain the vast majority of the packets and therefore consume most of the sampling resource. Since sampled packets will be processed by a flow table, which usually resides in relatively slower DRAM, to generate flow records like in Cisco's Netflow, only very low packet sampling rate (say 1/500) can be supported on high speed links. Such low sampling rate implies that the sampled flow records capture most of the large flows, but miss a majority of small and medium flows. This information loss on small and medium flows significantly affects the accuracy of the estimation of various network statistics such as flow size distribution and per-flow traffic, over the sampled traffic. With uniform packet sampling, it is hard to improve this situation, since even when we increase the sampling rate, most of the additional updates hit the large flows, doing little to increase the number of small and medium-sized flows.

We realize that if we are able to sample packets in small and medium flows at much higher rate than those in large flows, we will be able to gather richer and more useful information than uniform sampling. This is not hard to do if we *somehow* know, when a packet pkt arrives, the current size s of the flow it belongs to; We can simply decrease the sampling rate of its packets when a flow grows larger. In other words, we make the sampling rate of a packet pkt a decreasing function (denoted as f) of its current flow size s. This way, we can significantly increase the packet sampling rate of the small and medium flows at slight expense of the large flows[3], again thanks to the Zipfian nature of the Internet traffic. For example, suppose the largest 1% of the flows contain 90% of the Internet traffic. Then reducing the packet sampling rates of the top 1% flows by 12.5% will save us enough sampling resource to increase the sampling rate of the rest of the flows by 100%. *One key contribution of this work is to develop a sampling theory as to how to determine the right sampling function $f(s)$ so that the resulting sample allows for accurate estimation of various network statistics under a fixed sampling resource constraint.*[4]

Now the question is how do we know, when a packet arrives, the exact size of the flow that the packet belongs to. Such exact information is available only if we keep per-flow information for every flow and update the corresponding flow entry for every incoming packet, but as we mentioned before, this is not practical on high speed links. We develop a novel packet sampling methodology called "sketch-guided sampling" that circumvents this problem. Our main idea is to use a small (lossy) synopsis data structure (called a counting *sketch*) to encode the approximate sizes of all flows. This data structure is small enough to fit in fast SRAM so that it is able to process all the packets (without any sampling). For each incoming packet pkt, we first look up the data structure to determine the approximate size $\hat{s}$ of the flow it belongs to, and then sample it with probability $f(\hat{s})$. The estimation error of $\hat{s}$ may impact the precision of estimates based on our sampling theory, but the estimates remain unbiased. Our evaluation shows that our revised sampling theory based on $f(\hat{s})$ works almost as well as the prefect case based on $f(s)$.

Our SGS methodology will allow us to achieve more accurate estimations of many different network statistics than uniform sampling. In this work, we focus on three of them: (a) per flow usage accounting, (b)identification of the set of medium to large flows and estimation of their size, and (c) estimation of the distribution of flow sizes in arbitrary subpopulations of the traffic [5]. Our choice is based on the observation that uniform sampling leads to poor estimation of these network statistics. The problem and applications of estimating each of these three statistics have been studied independently and solutions based on sampling and data streaming techniques have been proposed. In this work, our focus is on demonstrating the utility of the SGS methodology and evaluating its impact on the accuracy of existing techniques.

The rest of this paper is organized as follows. Section II provides a brief survey of the work on sampling techniques with application to network monitoring, and

[3]Note that the relative error of our estimates of the large flows won't suffer significantly since the denominator is large.

[4]Note that we are not the first to propose the concept of size-dependent sampling. Our work differs from existing work in that it is the only work that allows for the sampling decision to be made online; all other sampling work makes decision off-line on the flow records that have already been collected, with or without sampling, which has more information available to them and therefore is able to achieve stronger properties.

identifies our contributions in relation to this work. Section III gives a high-level description of the proposed mechanism and develops the mathematical machinery used in our design of various sampling functions, followed by a description of the counting sketch (streaming data structure) we propose to use for maintaining approximate per-flow state. Section IV develops three specific applications based on the proposed framework and demonstrates resulting improvements in estimation accuracy. We conclude in Section V.

II. Background and related work

Solutions for network monitoring using data from a sample of packets typically use the following architecture. A monitoring point tapping into the live packet stream uses a selection process to pick a small sample of these packets and hands them off to a reporting process that aggregates them, typically into flow records, and exports these records using an information export protocol. An independent monitoring station collects this exported information and makes it available to various applications that use this sampled data to infer specific characteristics of the monitored traffic.

The Packet Sampling (PSAMP) working group at IETF is chartered to define a standard set of capabilities for network elements to sample subsets of packets by statistical and other methods [6], [7], [8]. Similarly, the IP Flow Information Export (IPFIX) working group has the specific goals of defining the notion of a "standard IP flow", devising data encodings that support analysis at various levels of aggregation, and considering the notion of IP flow information export based upon packet sampling [9], [10], [11]. The selection process in this architecture, as defined by the PSAMP working group, provides a detailed description of various sampling and filtering techniques that can be used to decide which packets are selected for reporting. The IPFIX working group specifies a detailed architecture for the reporting process that contains rules for encoding the observed packets into *Flow Records* and packetizing the selected flow records into IPFIX messages that are then exported to a *Collector*.

In their work on *Building a Better NetFlow*, Estan et al. [12] identify the major shortcomings of NetFlow, which include the large variability in the number of flows, and the mismatch between flow termination heuristics and the analysis. They propose the use of epoch-based termination of flows and adaptive sampling to overcome these problems. Their work, however, does not fundamentally change the nature of this uniform (or near-uniform) packet sampling in NetFlow. Hohn and Veitch [13] 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 idea of size-dependent flow sampling has been proposed and studied from various perspectives by Duffield, Lund and Thorup [14], [15], [16]. They in general solve the following problem. Suppose the monitoring device has collected a set of NetFlow-like flow records, and after a certain period of time, due to the storage constraint, only a small percentage of such records can be kept. Then what records should be kept and what should be discarded? The solutions proposed in [14], [15], [16] are to sample (and keep) flows of different sizes with different probability. Clearly different variants of solutions are called for under difference resource constraints and operation conditions. In [14], they study how to sample from (unsampled) NetFlow records so that estimations made on the sampled flow records are unbiased and have small MSE, under soft and hard resource constraints respectively. This is extended in [15] to the case of sampling from NetFlow records constructed from uniformly sampled packets. Our work is fundamentally different from all their approaches in that we need to make an online decision on whether or not to sample a packet while their approach makes an off-line decision on whether or not to sample a flow record that has already been collected and stored. In addition, since we are performing packet sampling and their works are performing flow sampling, our sampling objectives and corresponding mathematical analysis are fundamentally different from theirs.

Flow sampling is used for sampling the same set of flows (i.e., consistently) across multiple network elements [13]. In flow sampling, a flow is chosen uniformly at random with a probability p. All packets in the sampled flow will be sampled. Flow sampling can be implemented using a hash function by sampling all flows whose flow label (source and destination IP addresses and port numbers) is hashed to a particular set of values. Flow sampling is shown to be a better alternative to packet sampling for estimating certain network statistics such as flow size distribution [13]. However, sampling a flow with probability p in general cannot guarantee a reduction of factor $1/p$ at any time scale, because at the large time scale elephant flows may be disproportionally sampled, and at the small time scale bursty flows that consume the majority of link rate may be disproportionally sampled. This makes flow sampling unsuitable for applications that operate under a hard resource constraint (as in [16]).

Several mechanisms that use *sketches* to monitor specific properties of the network traffic have been

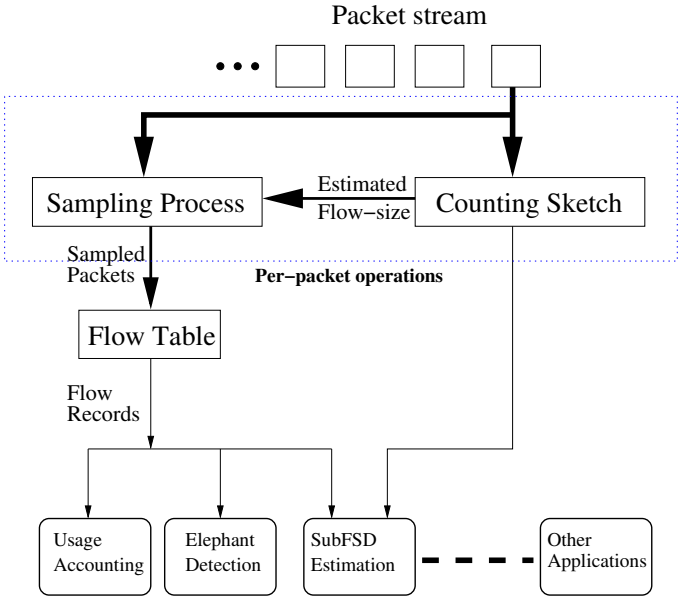


Fig. 1. Architecture for using sketches to guide packet sampling.

proposed recently. Sketches are extremely simple data-structures that maintain approximate information and can be updated at line speeds. Examples where sketches have been used to guide the sampling decision for individual packets include a multistage filter to detect large flows, designed by Estan and Varghese [17] and an automated worm fingerprinting mechanism that classifies content segments that are seen to be coming from multiple sources and going to multiple destinations as possible worm signatures [18]. Such solutions typically use a sketch as a high-speed counting device, and select packets with dominant counts in such sketches for further processing. These form a starting point for the streaming guided sampling techniques that we explore in this paper.

III. Design of Sketch-Guided Sampling

As we described before, uniform sampling is not accurate and efficient for the estimation of various network statistics because it collects the vast majority of samples from the few large flows while skipping most of the small and medium flows. In this section, we describe our sketch-guided sampling (SGS) scheme that aims at improving the sampling rate of small and medium flows at slight expense of large flows. The basic idea of SGS at the high level is quite simple and its overall architecture is shown in Figure 1. Upon the arrival of each packet, the SGS scheme tries to determine the current size of the flow that the packet belongs to, from the counting sketch module. The estimated flow size $\hat{s}$ is fed to the sampling process, which will sample the packet with probability $f(\hat{s})$. The sampled packets will be used to update the flow table. Here f is the aforementioned (size-dependent) sampling function. The (sampled) flow records and sometimes the counting sketch will be used for estimating various network statistics such as per-flow traffic, identification of large flows, and estimating the subpopulation flow size distribution (SubFSD).

As we discussed earlier, intuitively, f should be a decreasing function of s so that higher sampling rates can be applied to small and medium flows, but what kind of function should f be exactly? In Sec. III-A, we develop our sampling theory that develops a family of f that can be useful for the accurate estimation of various network statistics, assuming the perfect estimation of the current flow size s by the counting sketch. We also study how to design a robust f such that the estimation error on the flow size s will not impact the statistics estimated from the sampled packets significantly. Then in Sec. III-B, we describe our design of our counting sketch that can operate at full line speed and provide fairly accurate flow size estimation to the sampling module.

A. Sampling theory behind the SGS design

In this section, we derive a family of sampling functions, based on which the sampling module can collect packet samples that possess certain desirable statistical properties. We motivate this study by first examining the statistical properties of the packet samples collected by uniform sampling. Suppose in uniform sampling, each packet in a flow of size s is sampled with probability p. Then each sampled packet represents $1/p$ packets in the original stream, on the average. Thus, if the number of sampled packets is k, an unbiased estimator for the overall flow size is $\hat{s} = k/p$. The variance of this estimator can be shown to be $s(1-p)/p$, which is equal to its mean square error (MSE)[5] since $\hat{s}$ is unbiased. The *root mean square error* (RMSE) of this estimator is $\sqrt{(s(1-p)/p}$, which clearly grows much slower than the flow size s. In other words, the *relative* error in estimates reduces with rate $O(s^{-0.5})$ when the actual size of the flow s increases. While this might be desirable for certain applications, one can envisage others that would desire different growth functions of RMSE.

1) The constant relative error case: In many applications such as per-flow traffic accounting, we often would like the aforementioned RMSE to grow linearly with the actual flow size. In other words, we would like the average estimation error to be $\epsilon\, s$ (or $O(s)$) for a flow of size s, which corresponds exactly to constant relative error. Constant relative error is desirable

[5]The MSE of the estimator $\hat{X}$ of a parameter X is defined as $E[(\hat{X}-X)^2]$.

in traffic accounting because the average percentage of overcharge/undercharge remains the same irrespective of actual usage. We show next that this can be achieved by choosing a proper sampling function f.

We temporarily assume that our counting sketch provides perfect estimation of the flow sizes. Now given a flow of size s, recall that in our SGS scheme, its first, second, ..., and s_{th} packets will be sampled with probability $f(1)$, $f(2)$, ..., and $f(s)$ respectively. As discussed before, f is in general a decreasing function, i.e., $f(1) \geq f(2) \geq ... \geq f(s)$. Suppose k packets are sampled from a flow with probability p_1, p_2, ..., p_k respectively. Then the total number of packets in this flow can be estimated as $\hat{s} = \sum_{i=1}^{k} 1/p_i$. The RHS of the estimator is equivalent to $\sum_{i=1}^{s} u_i * 1/f(i)$, where u_i are mutually independent indicator random variables taking the value 1 if packet i is sampled and zero otherwise. The variance of the estimator $\hat{s}$ is given by the sum of the variance of the individual terms in the summation, due to their independence. Thus, $Var(\hat{s}) = \sum_{i=1}^{s} Var(u_i/f(i))$. Here, $Var(u_i/f(i)) = (1 - f(i))/f(i)$, which gives:

$$Var(\hat{s}) = \sum_{i=1}^{s} (1 - f(i))/f(i) \tag{1}$$

Now, using the property that $f(s) \leq f(i)$, $\forall i \leq s$, we can obtain a bound on the variance of the estimator as:

$$Var(\hat{s}) \leq s(1 - f(s))/f(s) \tag{2}$$

To make sure that RMSE grows at most linearly (i.e., relative error no more than a constant ϵ), we can simply make the RHS of Eqn. 2 equal to ϵs, which gives $f(s) = 1/(1 + \epsilon^2 s)$.

Readers may feel that the relaxation in equation 2 is unnecessary. We can make the variance (MSE) of $\hat{s}$, which is $\sum_{i=1}^{s} (1 - g(i))/g(i)$ when using g as the sampling function, exactly equal to $(\epsilon s)^2$. One can easily verify that this can be achieved by using the following sampling function: $g(i) = \frac{h(i)}{h(i)+1}$ where $h(i) = \epsilon^2(2i - 1)$, for $i = 1, 2, ..., s$. This optimization however is not recommended for two reasons. First, here we are assuming the perfect estimation of flow size by the counting sketch, which in reality will have estimation error. We discover that its impact on the sampling operation using g is much higher than that using f because the aforementioned relaxation in f provides robustness against this estimation error. Second, suppose that the probabilities with which the k sampled packets were sampled are $p_1 \geq p_2 \geq ... \geq p_k$. Some applications (e.g., SubFSD, to be discussed in Sec. IV-C) can only make accurate inferences on a packet sample in which each packet is sampled with equal probability, which can be achieved by resampling these packets. In this case, the maximum sampling rate we can achieve after resampling is p_k. Using sampling function f guarantees that the packet sample after resampling will not have RMSE larger than ϵs, but using g cannot guarantee that. Therefore, although for the purpose of per-flow traffic accounting it makes sense to use g, we adopt function f because it is applicable to a much wider spectrum of applications.

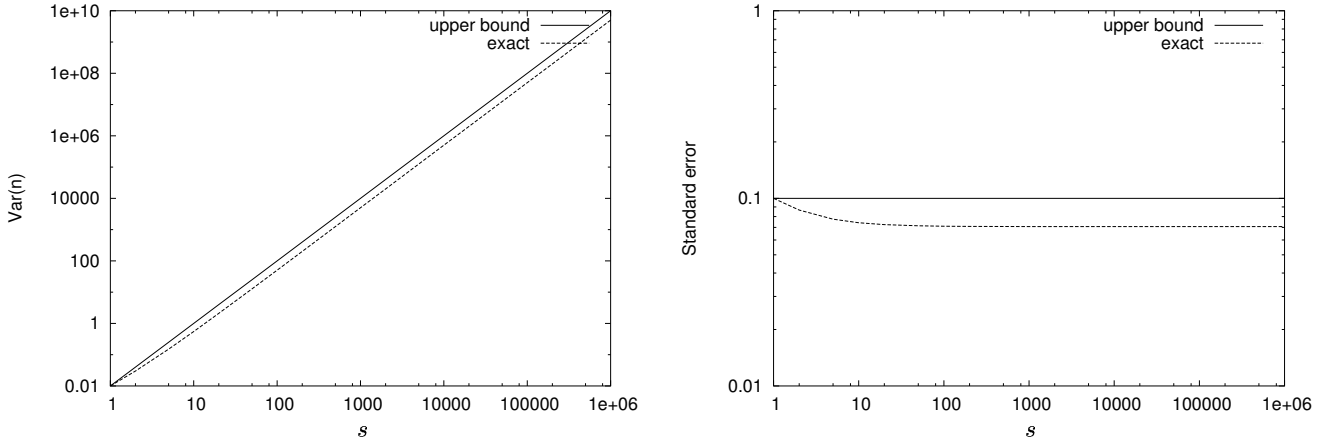
2) Generalizing to $O(s^\alpha)$ RMSE: We have just developed the sampling theory for achieving linear RMSE (i.e., $O(s)$ RMSE). In some other applications, however, one may want to have RMSE of the form $O(s^\alpha)$ where $1/2 \leq \alpha \leq 1$ is a tunable constant. Note that the aforementioned uniform sampling and linear relative error corresponds to the case $\alpha = 1/2$ and $\alpha = 1$ respectively. To make RMSE grow with rate $O(s^\alpha)$, we let $\sqrt{(s(1 - f(s))/f(s)}$ be equal to $\epsilon\, s^\alpha$ for $s = 1, 2, ...$, which gives $f(s) = 1/(1 + \epsilon^2 s^{(2\alpha-1)})$. For $\alpha = 1/2$, we have $f(s) = \frac{1}{\epsilon^2+1}$ which is a constant as expected, since this is the case of uniform sampling.

The bound in equation 2 is tight for $\alpha = 1/2$ (the case of uniform sampling). For $\alpha = 1$, which corresponds to the case of constant relative error, the upper bound given by equation 2 is not tight for all values of n. Figure 2 shows the variance and standard error of guided sampling, with $\alpha = 1$ and $\epsilon = 0.1$. The upper bound and the exact values are computed using equations 1 and 2. Figure 2(b) shows that the upper bound on standard error given by equation 2 is about 30% larger than the exact value of the standard error which can be calculated from equation 1. This slack is due to the approximation used to get rid of the summation in equation 2.

B. Implementation using Sketches

Our sampling theory establishes a family of guided sampling algorithms where the probability with which a packet is sampled depends on the size of the corresponding flow at that point of time. However, maintaining exact information about every flow seen at a monitoring point is prohibitively expensive at high speed. We would like to relax this requirement of knowing the exact flow size, trading it for the knowledge of an approximate estimate that might be relatively easier to obtain. We now present such a solution that uses a synopsis data structure from a family known as *couting sketches* to maintain approximate estimates of per-flow size.

Since this counting sketch needs to process each and every packet, for it to operate at link speeds such as OC-192 and OC-768, each operation has to be performed



(a) Variance of estimates.

(b) Standard error.

Fig. 2. Variance and standard error of guided sampling, with $\alpha = 1$ and $\epsilon = 0.1$.

within tens of nanoseconds. After evaluating a number of possible candidates, we choose a very simple synopsis data structure – an array of counters indexed by a hash fuction – to track per-flow sizes approximately. The primary reasons for picking this solution over more sophisticated alternatives such as the Space-Code Bloom Filter [19] were its extreme simplicity and efficiency that allow us to support very high links [20], [5].

Every counter in this array is initiated to 0 at the beginning of a measurement epoch. The update and estimation operations at each packet arrival are performed as follows. Upon arrival of a packet at the measurement point, its flow label[6] is hashed to generate an index into this array, and the counter at this index is incremented by 1. The updated value of the counter is passed on to the selection process as the estimated size of the corresponding flow. Collisions due to hashing might cause two or more flow labels to be hashed to the same index. The value of the counter at such indices would represent the total number of packets belonging to all of the flows hashing to the 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, but introduces some estimation errors as discussed in the next section. Efficient implementations of hash functions [21] allow the online streaming module to operate at speeds as high as OC-768 without missing any packets.

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

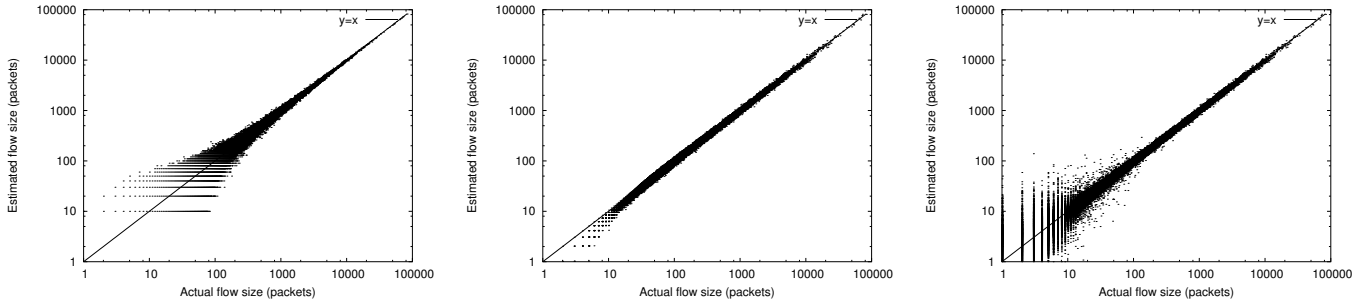
This sketch data structure can be implemented using fast memory (SRAM) to keep up with high packet arrival rates. A naive implementation would simply place an array of 32-bit counters in SRAM. This should be practical for arrays of up to a million counters using 4MB of commodity SRAM. Larger arrays can still be implemented using the techniques for efficient implementation of an array of counters, proposed by Ramabhadran and Varghese [22]. The computational cost of each update is one hash function computation followed by one read and write to SRAM. The estimate of flow size is nothing but the updated counter value, available at no extra computational cost. We refer the reader to [20] for a more detailed analysis of the computational and storage complexities of this sketch.

IV. Applications

In this section, we apply our SGS methodology to the estimation of three types of network traffic statistics. They are: per flow usage accounting, described in section IV-A; identification of the set of medium to large flows and estimation of their size, described in section IV-B, and estimation of the distribution of flow sizes in arbitrary subpopulations of the monitored traffic, described in IV-C.

A. Usage accounting

Accounting for bandwidth usage is perhaps the most basic application of sampling in network monitoring. Per-flow traffic accounting has applications in usage-based charging/pricing, network anomaly detection, security, network planning, peering policy, customer acquisition and traffic engineering [15], [17]. Previous work



(a) Uniform sampling, $p = 1/10$.

(b) Guided sampling, $\epsilon = 0.1$, exact information.

(c) Sketch Guided sampling, $\epsilon = 0.1$, approximate information.

Fig. 3. Actual and estimated flow sizes using different sampling techniques.

on this application has focussed on the use of sampling at the collector of flow records. Specifically, previous work by Duffield et al. [14] assumes that the initial selection and reporting processes can process each packet, exporting flow records to a collector which samples and archives only a subset of these records to save storage space. Duffield et al. show that uniform sampling of flow records is not the best strategy, due to the heavy tailed distribution of flow sizes, and propose a size-dependent sampling scheme that gives priority to larger flows. In subsequent work [15], Duffield et al. quantify the impact of uniform packet sampling on the accuracy of usage estimates in an operational measurement infrastructure. They show that if packets are sampled uniformly with probability p, then $\hat{s} = k/p$ is an unbiased estimator of the total number of packets in a flow, where k is the number of samples from the flow. They also show that the *standard error*[7], or relative RMSE, defined as $\sqrt{Var\ \hat{s}}/s$, for this estimator is $\sqrt{(1-p)/ps}$. Thus, the standard error of estimates based on uniform sampling reduces as the flow size s increases. This is shown in figure 2(b) in the curve for $\alpha = 1/2$.

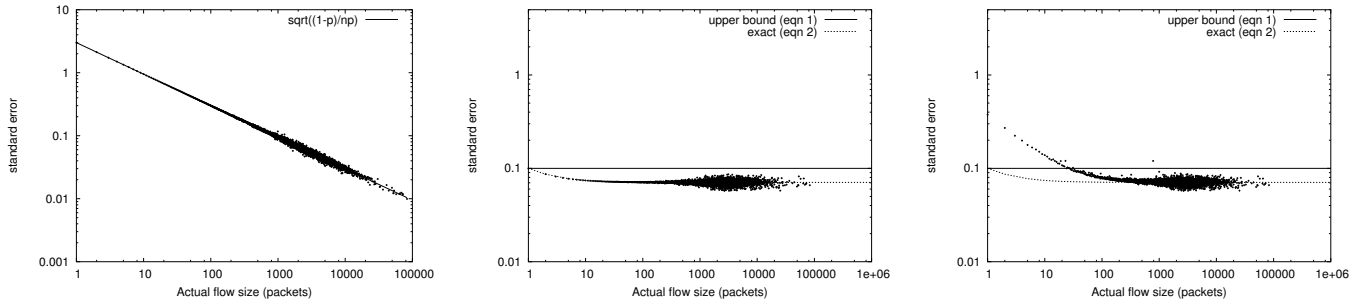
A property of uniform sampling is that most of the samples are collected from a small number of large flows with increasing estimation accuracy for such flows. However, because the sampling "budget" is limited due to system resource constraints, the additional accuracy for large flows may be a waste of this budget. In other words, the downward sloping shape of the curve for standard error (figure 2(b)) is fixed in uniform sampling, while applications might prefer other shapes such as a flat curve. The use of SGS provides this flexibility to network monitoring applications. In particular, the guided sampling with $\alpha = 1$ provides flat standard error in estimation irrespective of the original flow size, as shown in figure 2(b).

Figure 3 shows the usage estimates generated from samples collected via various techniques from the same trace. The trace used in this and all the subsequent experiments was collected at the 1 Gbps access link connecting the UNC campus to the rest of the Internet on Thursday, April 24, 2003 at 11:00 am. This trace contains 27,507,496 packet headers from 1,133,150 flows constituting 500 seconds of traffic on the link. We obtained similar experimental results over publicly available traces from NLANR and USC that we omit in the interest of brevity.

As is common with Internet traffic, the flow size distribution in the trace used in Figure 3 is heavy-tailed with a large number of small flows and a few large flows. Figure 3(a) shows estimates of flow-size derived from uniform packet sampling with probability 1/10. The estimation accuracy is very low for small flows, and is low for medium sized flows with 10 to 1000 packets too, but improves significantly for the very large flows. On the other hand, guided sampling with exact flow-size information provides a constant relative accuracy, as demonstrated by the clustering of all points in a narrow band along the $y = x$ line in figure 3(b). Due to the use of log-scale on both axes, this narrow band parallel to the $y = x$ line represents a region linear in size with respect to s, the value of the actual flow size plotted on the x axis. The slight broadening of this band near $x = 1$ is due to the presence of an extremely large number of small flows in the trace, which ensures that even the extremely rare estimation-errors are represented by a point.

Replacing the assumption of knowledge of exact flow-size in guided sampling with sketch-based approximate estimates in SGS results in some loss of accuracy, as shown in figure 3(c). The additional inaccuracy is mostly

[7]We use the terms *standard error* and *relative RMSE* interchangeably.



(a) Uniform sampling, $p = 1/10$.

(b) Guided sampling, $\epsilon = 0.1$, exact information.

(c) Sketch Guided sampling, $\epsilon = 0.1$, approximate information.

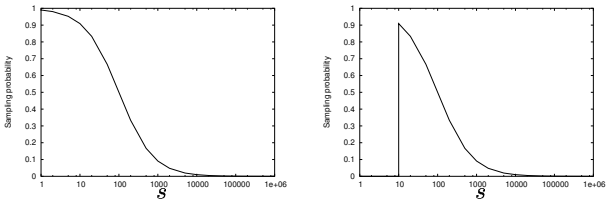
Fig. 4. Experimentally determined standard error and analytical values using different sampling techniques.

limited to small flows. This inaccuracy comes from hashing collisions in the counting sketch that result in the generation of inaccurate flow-size estimates by the sketch. Notice that collisions in hashing can only cause over-estimation of the sizes of the colliding flows. However, since SGS uses this estimate only in determining the sampling rate, the estimation through SGS remains unbiased. The over-estimation of flow-sizes by the sketch does force SGS to pick a lower sampling rate, resulting in higher standard error. This impact is muted for all flows of medium and large sizes because SGS uses low sampling rates for such flows anyway, and the additional reduction in this sampling rate due to over-estimation of flow-sizes by the sketch has negligible impact.

The standard error of estimation using uniform sampling, guided sampling with exact information and sketch guided sampling using approximate information is shown in figure 4. To generate sufficient number of points, the estimation experiments were repeated 100 times with different seeds. The experimental values of standard error for uniform sampling and guided sampling with exact information faithfully follow the respective analytical curves. On the other hand, the experimentally determined points for sketch guided sampling lie above the analytical upper bound for guided sampling with exact flow-size information (figure 4(c)). This increase in error is due to inaccuracies in the estimates of flow-size that are provided by the counting sketch. However, this inaccuracies have an impact only on the estimation of very small flows, and even in that region, the overall accuracy is much better than that provided by estimation based on uniform sampling.

B. Detecting Elephants

The previous example application, per-flow usage monitoring, is closely related to our second example



(a) Usage accounting

(b) Elephant detection

Fig. 5. Sampling probabilities as a function of flow size for two different applications.

application, using SGS to detect large flows (a.k.a. *heavy hitters* or *elephants*). The ability to detect large flows and estimate their sizes has applications similar to those of usage monitoring, such as anomaly detection, verification of compliance with service level agreements and traffic engineering. In monitoring very high speed links, it is often necessary to ignore the small flows (or mice) and focus on the medium to large size flows, since it significantly reduces the number of flows that need to be tracked, freeing up resources for tracking the larger flows. Due to the Zipfian nature of the distribution of flow sizes in the Internet traffic, ignoring small flows in fact will allows us to track all the medium and large flows with much higher accuracy than attainable while tracking all flows.

Our SGS-based solution for estimating the sizes of medium and large flows (called "elephant detection") is an adaption of the solution for per-flow usage accounting. The crucial difference is that for elephant detection, we can ignore the small flows. This is implemented by setting a threshold T and modify the sampling function used in usage accounting $\left(f(s) = 1/(1+\epsilon^2 s)\right)$ in such a way that $f(s)$ becomes zero for $s < T$, but remains

the same (as in usage accounting) for $s > T$. The flow size estimator is modified accordingly by incrementing the non-zero estimates of flow size by the threshold T.

Figure 5 shows the sampling rates as a function of the flow size for the two applications of usage accounting and elephant detection, respectively. The parameters used in both figures are $\alpha = 1, \epsilon = 0.1$ and the threshold T is set to 10 packets for the curve in figure 5(b). The area below the curves in figure 5 indicates the number of samples collected for the corresponding applications. The total number of samples collected is the product of this value and the actual flow size distribution (histogram of the flow sizes) which is known to be heavy tailed in the case of Internet traffic. By excluding the flows whose sizes are below the threshold T, our solution collects a smaller number of samples from a much smaller number of flows. If desired, the saved resources can be used to increase the sampling rates for the medium and large flows, resulting in much higher estimation accuracy.

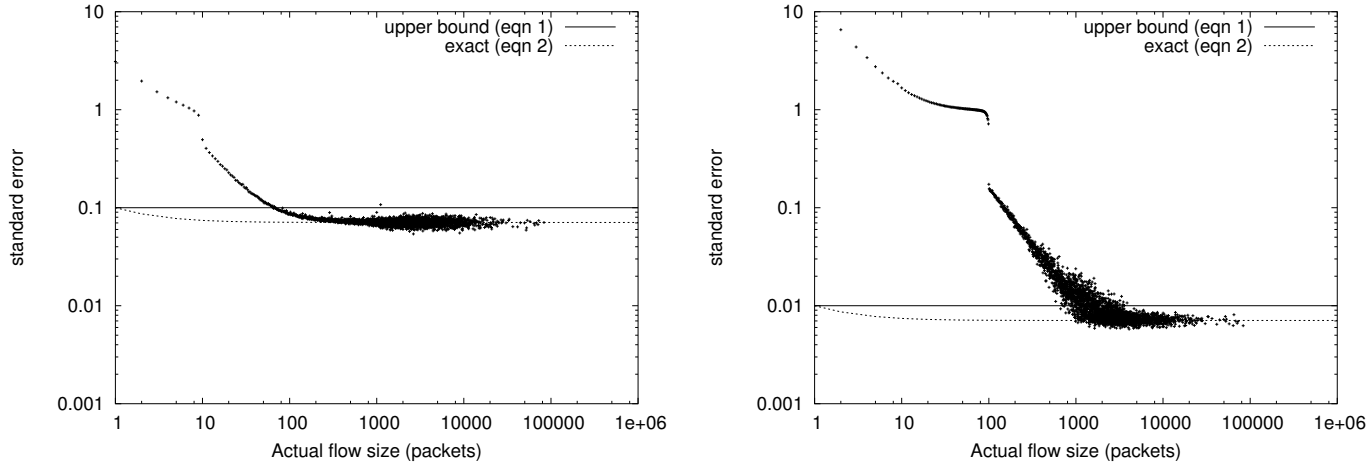
Although the additional inaccuracy introduced by ignoring the first few packets of the flow while its size is below the threshold can be substantial for flows whose sizes are just above the threshold T, this effect diminishes rapidly and is negligible for flows larger than $10T$. This is best demonstrated through the experimental results shown in figure 6. The analytical curves ("upper bound" and "exact") are from the equations 1 and 2 in the previous section (with no cutoff before the threshold T). The errors in estimating the flow sizes are captured in the "black dust" curves. The high errors for flows smaller than the threshold can be attributed to the fact that our elephant detection mechanism is designed to avoid collecting samples from such flows. In practice some of these flows are still sampled due to overestimation of their sizes by the counting sketch (the case of hash collisions). There is a perceptible downward jump around the threshold T as flows suddenly begin to be sampled by the elephant detection mechanism. The accuracy of size estimations improves rapidly from this point on, until it approaches the analytical accuracy of usage accounting using guided sampling with exact flow size information (i.e., the ideal case).

Previous work by Estan and Varghese [17] for elephant detection used two alternative techniques for identifying potential elephants. Once a flow is identified as a potential elephant, an entry is created for it in fast memory (SRAM). Every packet that belongs to one of the identified potential elephants causes an update to the corresponding entry in fast memory. The two alternative techniques are *sample and hold*, where packets are sampled at a low uniform probability and an entry is created if one doesn't already exist for the corresponding flow, and *multistage filter* where multiple parallel count sketches with independent hash functions are used, and a flow is identified as a potential elephant if *all* the corresponding counters in the parallel sketches exceed some predetermined threshold value.

The main difference between our scheme and that by Estan and Varghese [17] lies in our use of guided sampling even for the flows whose sizes are above the threshold versus the deterministic per-packet update by the latter. Our use of guided sampling provides constant relative error, with the possibility of picking other functions from the α-family introduced in section III. The use of per-packet updates in [17] means that the only errors in estimation are due to the initial detection algorithm, resulting in slightly better accuracy than our scheme. However, in our scheme, the number of updates to the (elephant) flow table stored in fast memory can be much smaller because of the sampling operation. For example, for the largest flow in the UNC trace, which has 83,097 packets, per-packet update would require slightly less than 83,097 updates to the fast memory. Our SGS solution, on the other hand, only needs 670 updates, with $\epsilon = 0.1$. This corresponds to two orders of magnitude fewer updates to the corresponding flow entry in the fast memory. This reduction allows our scheme to scale to much higher link speeds (or allowing the use of slower and less expensive DRAM) for the following reason. In both schemes in [17], the flow entries have to be organized as a hash table in the fast memory to allow for per-packet updates. The cost of storing these flow entries and updating them is much higher than the counting sketch used in our scheme. The flow table in our SGS scheme, on the other hand, can be stored in inexpensive DRAM because the number of updates it needs to process is two orders of magnitude smaller. Therefore, our SGS scheme offers an attractive and less costly alternative to [17] for tracking medium to large size flows, at the slight cost of estimation accuracy.

C. Estimating the flow-size distribution

The final application of our SGS scheme is more accurate estimation of the Flow Size Distribution (FSD) of the Internet traffic. The 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. Furthermore, 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),



(a) $\epsilon = 0.1$, threshold=10.

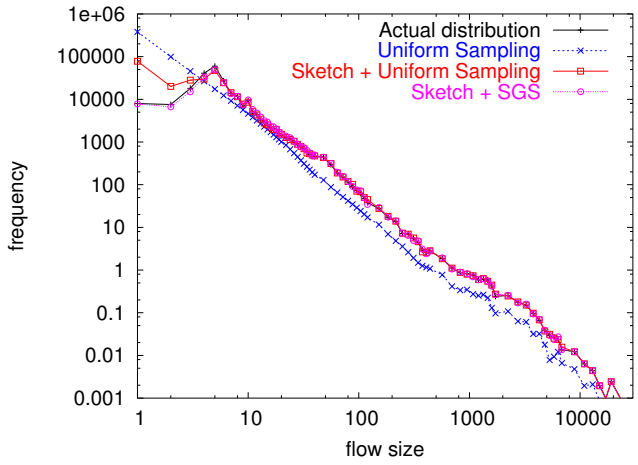
(b) $\epsilon = 0.01$, threshold=100.

Fig. 6. Experimentally determined standard error for elephant detection using Sketch Guided sampling with various parameters.

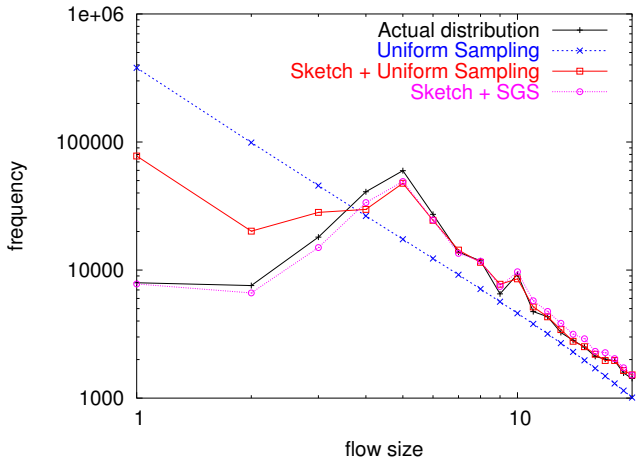
by a source or destination IP address/prefix, by protocol (e.g., TCP or UDP), or some combination thereof. For example, to investigate a sudden slowdown in DNS lookups, a network operator may specify all flows with source or destination port 53 as the subpopulation of interest and query the data collected in the preceding hour.

Accurate solutions for usage accounting on a per-flow basis provide a naive solution to the problem of estimating FSD – simply sort the estimated sizes of all flows to obtain a distribution. However, there are two reasons to not take this approach. First, one may wish to solve the easier problem of estimating the FSD without being forced to solve the harder problem of identifying all flows and estimating their sizes accurately. Second, by designing an independent solution for estimating the FSD, one might hope to achieve better estimation accuracy than that provided by the aforementioned naive solution. Indeed, previous work on estimating the FSD ([23], [13], [20], [5]) has shown that data collected through sampling or sketching can be used to estimate the FSD much more accurately than it can be used to estimate the size of individual flows. In particular, we have presented in [5] the design of an efficient mechanism to provide accurate estimates of the FSD for arbitrary subpopulations specified after the act of data collection. That solution uses two sources of data, a counting sketch which is the same as in section III-B and a variant of sampled NetFlow [4] that implements uniform packet sampling. It then uses an estimation algorithm to statistically correlate and decode both sources of data to accurately infer the flow size distribution for any subpopulation.

In this section we replace the data gathered from uniform sampling, as used in our previous solution [5], with data gathered by SGS, to further improve the accuracy of estimation of that solution. The estimation algorithm in [5] is an iterative Expectation Maximization (EM) algorithm that computes the Maximum Likelihood Estimate (MLE) of the distribution that would cause the observations as collected at the monitoring point. Here, the observations are the values in the count sketch (array of counters) and the samples collected in the sampling module. Without repeating the details of this algorithm, we would like to point out that it was originally designed to work with uniform sampling. Since the packets in a flow is not uniformly sampled in our SGS scheme, we need to resample them into a *locally uniform* (defined next) sample to work with this algorithm. The resampling algorithm is quite simple: among all packets that were sampled and hashed to counter k, pick the sampling rate q seen by the last *sampled* packet as the target sampling probability for counter k. Note that due to the monotonically non-decreasing nature of sampling probabilities, q is the smallest sampling probability seen by any of the sampled packets hashed to counter k. Suppose m packets (in their arrival orders) are sampled with probabilities p_1, p_2, ..., and $p_m = q$. Then our resampling process is simply to sample them with probability q/p_1, q/p_2, ..., q/p_{m-1} and 1 respectively. One can see that the net effect of this is that the resulting packet sample is statistically equivalent to uniform sampling of all packets hashed to counter k with probability q. Therefore, we refer to the resulting samples as *locally uniform*, where

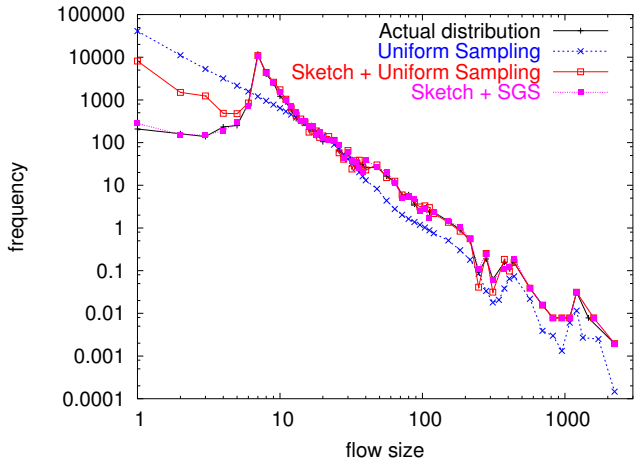


(a) Complete distribution.

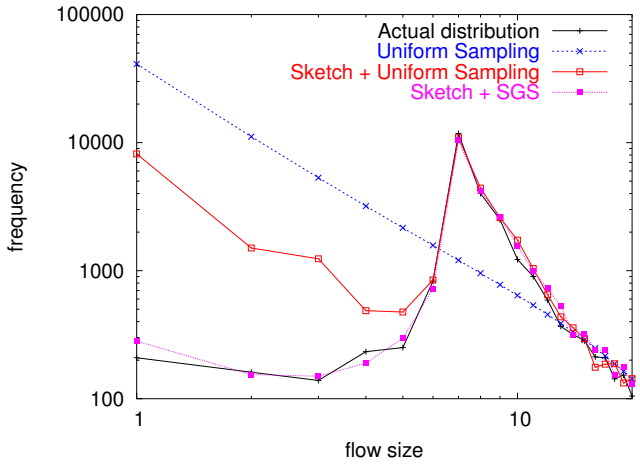


(b) Zoom in to show impact on small flows.

Fig. 7. Estimates of FSD of http flows using various data sources.



(a) Complete distribution.



(b) Zoom in to show impact on small flows.

Fig. 8. Estimates of FSD of https flows using various data sources.

the locality of interest is the set of flows hashed to the same counter. With this resampling, the data collected by SGS can be readily used by the aforementioned EM algorithm in [5].

Our SGS scheme significantly improves the estimation accuracy of subpopulation FSD for small flow sizes with less packet samples collected from the traffic. Figure 7 shows the estimates of FSD of all flows from or to port 80, which corresponds to the `HTTP` traffic, in the trace from UNC, computed using various methods. The actual distribution has a peak at 7 packets, which is completely missed by estimation using only the data from uniform sampling using Duffield et. al.'s method [23]. Our previously proposed algorithm [5] detects this peak using the count sketch and uniform sampling simultaneously. However, this estimate is also somewhat inaccurate for the small flows as can be seen from figure 7(b). Replacing the uniform sampling data with data generated through SGS provides the most accurate estimation for small flows without compromising the accuracy in any other region. Figure 8 shows similar improvements in accuracy for flows to or from port 443, which corresponds to `HTTPS` traffic. In both figures, the sampling rate for uniform sampling was $p = 0.1$. The parameters for SGS used in this experiment were $\alpha = 1, \epsilon = 0.5$. With these parameters, the number of

packets sampled in SGS, *before* the resampling described above, was roughly the same as those sampled using uniform sampling. After resampling as described above, the total number of sampled packets in the SGS dataset was actually 30% less than in the dataset for uniform sampling.

V. Conclusions

Packet sampling is a basic ingredient of the architecture of network monitoring solutions. Recognizing that uniform packet sampling might not provide the best utilization of scarce resources available for data collection at high speed network devices, we have developed a new sampling methodology that uses approximate information from a counting sketch to guide the sampling decision. This SGS methodology allows a designer to tailor the sampling rate as a function of flow size, providing greater flexibility in the allocation of resources. Through the development of three applications based on SGS data, we have demonstrated the advantages of using SGS in specific scenarios. In usage accounting, SGS allows for the design of estimation mechanisms that have constant accuracy irrespective of flow size. For elephant detection, it reduces the number of updates to high speed memory by up to two orders of magnitude by allowing the sampling rate to change with flow size, achieving the targeted accuracy while minimizing resource usage. Similarly, for the estimation of flow size distribution of arbitrary subpopulations of traffic, we have demonstrated that the use of SGS improves the accuracy of our previously proposed algorithm [5] while collecting the same number of samples as uniform sampling. We have also discussed the implication of changing the size of the array of counters used in the count sketch and changing the sampling function or its parameters in SGS. We believe, that these applications of SGS are the first step in an exploration of the wide range of possibilities that are offered by the ability to guide the sampling decision on a per-packet basis.

References

[1] 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, extended abstract appeared in Proc. ACM IMC 2003.

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

[3] K. C. Claffy, G. C. Polyzos, and H.-W. Braun, "Application of sampling methodologies to network traffic characterization," in *SIGCOMM '93: Conference proceedings on Communications architectures, protocols and applications*. ACM Press, 1993, pp. 194–203.

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

[5] A. Kumar, M. Sung, J. Xu, and E. W. Zegura, "A data streaming algorithms for estimating subpopulation flow size distribution," in *Proc. ACM SIGMETRICS*, June 2005.

[6] "http://www.ietf.org/html.charters/psamp-charter.html."

[7] D. Chiou, B. Claise, N. Duffield, A. Greenberg, M. Grossglauser, P. Marimuthu, and J. Rexford, "A framework for packet selection and reporting," draft-ietf-psamp-framework-10.txt.

[8] T. Zseby, M. Molina, N. Duffield, S. Niccolini, and F. Raspall, "Sampling and filtering techniques for ip packet selection," draft-ietf-psamp-sample-tech-06.txt.

[9] "http://www.ietf.org/html.charters/ipfix-charter.html."

[10] G. Sadasivan, N. Brownlee, B. Claise, and J. Quittek, "Architecture for ip flow information export," draft-ietf-ipfix-architecture-06.

[11] J. Quittek, S. Bryant, and J. Meyer, "Information model for ip flow information export," draft-ietf-ipfix-info-06.

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

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

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

[15] N. Duffield and C. Lund, "Predicting resource usage and estimation accuracy in an ip flow measurement collection infrastructure," in *Proc. ACM Internet Measurement Conference*, Miami Beach, FL, October 27-29 2003, pp. 179–191.

[16] N. Duffield, C. Lund, and M. Thorup, "Flow sampling under hard resource constraints," in *SIGMETRICS 2004/PERFORMANCE 2004: Proceedings of the joint international conference on Measurement and modeling of computer systems*. New York, NY, USA: ACM Press, 2004, pp. 85–96.

[17] C. Estan and G. Varghese, "New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice," *ACM Trans. Comput. Syst.*, vol. 21, no. 3, pp. 270–313, 2003.

[18] S. Singh, C. Estan, G. Varghese, , and S. Savage, "Automated worm fingerprinting," in *Proc. of USENIX OSDI*, 2004.

[19] 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.

[20] 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.

[21] 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.

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

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