A Resource-minimalist Flow Size Histogram Estimator

UMass Technical Report TR-29-2008

Bruno Ribeiro[1], Tao Ye[2], and Don Towsley[1]

[1]Computer Science Department
University of Massachusetts
Amherst, MA, 01003
{ribeiro, towsley}@cs.umass.edu

[2]Sprint
One Adrian Court
Burlingame, CA, 94010
Tao.Ye@sprint.com

ABSTRACT

The histogram of network flow sizes is an important yet difficult metric to estimate in network monitoring. It is important because it characterizes traffic compositions and is a crucial component of anomaly detection methods. It is difficult to estimate because of its high memory and computational requirements. Existing algorithms compute fine grained estimates for each flow size, i.e. 1, 2,... up to the maximum number observed over a finite time interval. Our approach instead relies on the insight that, while many applications require fine grained estimates of small flow sizes, i.e. $\{1, 2, \ldots, k\}$ with a small k, network operators are often only interested in coarse grained estimates of larger flow sizes. Thus, we propose an estimator that outputs a binned histogram of size distributions. Our estimator computes this histogram in $O(k^3 + \log W)$ operations, where W is the largest flow size of interest to the network operator, while requiring only a few bits of memory per measured flow. This translates into more than 4 fold memory savings and an exponential speedup in the estimator as compared to previous works, greatly increasing the possibility of performing online estimation inside a router.

1. INTRODUCTION

Measuring the histogram of network flow sizes has been the subject of a number of recent studies [2–5, 8]. Although Internet routers handle traffic on a packet-by-packet basis at the IP layer, the statistics of the underlying flows are vital to network operators. The histogram of flow sizes, i.e. the number of flows with i packets, is an important traffic metric that gives insights to network monitoring applications such as traffic profiling, and aids fast detection of security attacks. In this work we refer to this histogram as the *size histogram*.

Directly computing size histograms requires an often expensive first step of classifying packet traffic into flows. Because as many as half a billion flows arrive every hour at an Internet core router, this task is extremely resource intensive. Reducing the load by aggressively sampling packets at random [3] (without much side information) has been shown to produce inaccurate histogram estimates [8]. Recently a number of alternative solutions employ data streaming algorithms to solve the histogram problem with more accuracy [2,4,5]. Kumar et al. [5] proposed sketches to maintain approximate flow size counts in an array of counters, then used a computationally intensive Expectation Maximization (EM) algorithm to compute the estimation from these counters. The authors advise network operators to use at least one 32 bit counter *per* flow in average. We argue that although this EM algorithm yields precise estimates, it is possible to obtain useful estimates much faster and with much smaller counters. Cormode et al. [2] computes the histogram using a sketch that maintains a tuple (flow size counter, flow id) in memory. It relies on flow filtering (thinning) to compute histogram estimates. The drawback of this method comes from its high memory requirements, which can be as high as dozens of bytes *per* flow. Lu et al. [6] uses a streaming algorithm tailored specifically to measure heavy tailed distributions that requires little memory space. Their estimator however needs to compute all flow sizes and use (even in a best-case scenario) $O(W)$ time, where W is the largest flow size of interest to the network operator.

In this work we propose an approach similar to Kumar et al. [5], using a sketch of flow size counters and a direct estimator. However, we trade-off flow size granularity for an exponentially faster estimator and less than 1/4 of memory usage (7 bits *per* flow) compared to [5]. In contrast to Lu et al. [6], our approach is exponentially faster and does not assume we know the distribution shape.

Reducing memory consumption. We are motivated by the following insight of traffic monitoring applications. Since size histogram provides a general overview of the traffic, rather than a direct measurement of usage, typical monitoring applications do not need fine grained counts of all flow sizes. In security event detection, we are interested in very small flow sizes (mice) such as 1, 2,

but only up to a certain value k. To measure the impact of medium and elephants flows, these larger flow sizes can be estimated in a binned fashion. Therefore we propose a new multi-resolution algorithm to estimate the size histogram with aggregated and probabilistic counting of large flows, and fine-grained counting for flow sizes up to k packets. As an example, for $k = 16$, we maintain per flow counters for each flow sizes 1,..,16, but flows of sizes from 17 packets to 32 packets, 33 to 64, etc, are counted probabilistically and estimated together. This allows for a faster estimator with reduced counter sizes, while still maintaining good accuracy. Using our approach with 6 bit counters and $k = 16$ we can probabilistically count flows of sizes up to $W \approx 10^{14}$.

Faster estimator. We also provide a simple estimator that is almost as accurate as a slower maximum likelihood estimator for small flow sizes. For larger flow sizes the second part of our technique estimates all histogram bins in $O(\log W)$ time, where W is the largest flow size of interest for the network operator. This is achieved by designing a space efficient low collision sketch. Our sketch divides and folds (multiplexes) Z virtual sketches into the physical space of one sketch. The low flow collision probability allows us to obtain good histogram estimates with $O(k^3 + \log W)$ operations in total. Note that W and k can be made as small as the network operator wants with no loss in accuracy. When faced with very high speed links and relatively low computing resources for monitoring and statistics gathering, our simple resource minimalist design makes a strong case for an in-line inside the router implementation.

The rest of the paper is organized as follows. Section 2 provides an overview of the algorithm. Section 3 illustrates data structure design and our estimator in details. Experiment results using trace data are shown in Section 4. We conclude with Section 5.

2. OVERVIEW

We follow a common design of breaking down the packet stream by measurement epochs. In each epoch our algorithm contains two phases: upon each packet arrival a data structure to count flow sizes is updated, at the end of the epoch we use an estimator to compute the histogram from this data structure.

The data structure works as follows. Each newly arrived packet is used to update a counter through a hash function, where all packets in a flow hash to the same counter. We keep M of such counters in a vector that we refer as the *sketch*. We choose a *universal hash function*, where a randomized algorithm is used to generate hash values for distinct flows. Hence, the collision probability of different flows hashed to the same counter is a simple function of the number of flows divided by the size of the sketch. We increase counter values probabilistically, in a variation of the approach in Morris [7]. Counters are

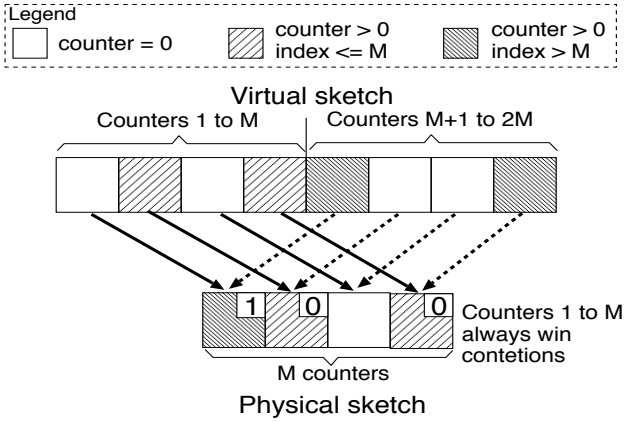

Figure 1: Multiplexing a sketch. One extra bit is used to store ownership in the physical sketch. Note that counters with index $\leq M$ in the virtual sketch always win contentions against counters with index $> M$.

incremented by 1 with probability 1 if the counter value, C, is less than k (a small constant defined by the network operator). Otherwise, if $C \geq k$ we increment the counter with probability 2^{-C+k-1}. This translates into grouping medium to large flow sizes into histogram bins $B_m = [k+2^m-1, k+2^{m+1}-1]$, $m = 0, \ldots, \log_2 W - 1$, where W is the largest flow size as defined by the network operator. As a result of this binning, we only need to use tiny counters (our experiments use 7 bit counters) as compared to other schemes, drastically lowering memory requirements. Note that this approach is performed entirely in software and does not require specialized hardware. In Section 3 we explain the details and also present a practical and efficient way to emulate a probabilistic counter without resorting to (slow) pseudo-random number generators.

At each sketch update our algorithm also updates a histogram of the sketch values. In the estimation phase, our estimator uses this sketch histogram to estimate the size histogram. It is shown in [5] that to estimate a flow of size i we need at least $O(i^3)$ operations to untangle the corresponding hash collisions. This results in a prohibitively high CPU cost with large maximum flow sizes, $W \gg 1$. Thus a fast estimator depends on a small fraction of hash collisions. Fortunately, universal hashing allows us to reduce the collision probability by simply increasing the sketch size, giving us the opportunity to reduce the estimator complexity.

However, a large sketch size is not desirable due to the corresponding increase in memory requirements. We therefore arrive at the sketch design shown in Figure 1 where the number of virtual sketches $Z = 2$. The idea is to use a virtual sketch that physically occupies half of its virtual size. Two virtual sketch counters share the same

physical sketch counter if both counters are zero. If one or both virtual counters are not zero we use the contention resolution algorithm described in Section 3.1.1. Our contention resolution has an overhead of $\lceil \log Z \rceil$ bits *per* physical counter. Let's look at an example on contention in Figure 1. Counters with indexes M and $2M$ in the virtual sketch contend for the same physical counter. Virtual counter M wins and virtual counter $2M$ is evicted from the virtual sketch. In Section 3.1.1 we see that virtual counter eviction is equivalent to flow thinning. In our experiments in Section 4 at most 15% of the flows are discarded due to counter eviction. Extra memory space could be used to store these evicted counters if flow thinning must be avoided. It is important to note that the extra CPU overhead using this approach when compared to a regular sketch is negligible.

Small counters with their exponential histogram bin sizes and small hash collision probabilities allow us to propose a $O(k^3 + \log W)$ histogram estimator. Note that k is a small constant typically $k \ll W$ (in our experiments we use $k = 16$ and $W = 10^{14}$). In what follows we detail our sketch data structure and simple estimator.

3. ALGORITHM

Our algorithm is made of two components, the sketch data structure and the estimator. We first describe the data structure used to sketch flow sizes based on each packet, then we describe our estimator that summarizes the distribution from the sketch.

3.1 Data structures

Our sketch data structure consists of three structures:

1. Sketch: This is the main structure that holds a sketch of flows and its sizes. Each packet is hashed into the sketch to increment its flow size count. There are M b-bit counters labeled $[c_m]$, $m = 1, \ldots, M$, plus one ownership bit *per* counter (in our experiments $b = 6$).

2. Sketch histogram: $g = (g_0, \ldots, g_{2^b-1})$ is the normalized histogram of the above counter values; it is updated upon changes of the ownership bits.

3. Pseudo-random auxiliary counters: We use $O(\log W)$ auxiliary counters to implement random sampling for flow counters in the sketch.

Here W is the maximum flow size of interest and

$$b \geq \lceil \log_2(\log_2 W + 1 + k) \rceil.$$

In what follows we illustrate the role of each of these in our algorithm.

3.1.1 Sketch

Our sketch is a virtual sketch with ZM counters, $Z \in \{2, 3, \ldots\}$, occupying the physical space of a sketch with M counters. We refer to this virtual sketch as a *Z-fold virtual sketch* or just a *virtual sketch* if the value of Z is clear from the context. Counters in the physical sketch are indexed from 1 to M. A counter in the physical sketch is shared by Z virtual sketch counters; each physical sketch counter has $\lceil \log_2 Z \rceil$ ownership bits. We call counters in the virtual sketch *virtual counters* or just *counters*. We call counters in the physical sketch *physical counters*. In what follows we consider $Z = 2$ to simplify our exposition. Let's follow the example shown in Figure 1. Virtual counters with indices c and $M + c$, $c = 1, \ldots, M$, are mapped into the physical counter with index c. A physical counter value represents the value of a virtual counter with index $\leq M$ if its ownership bit is zero. Otherwise it represents a virtual counter with index $> M$. Physical counters are initialized with value zero and with ownership bits set to one. Packets of a flow assigned to a virtual counter with index $> M$ will not change its corresponding physical counter if the physical counter has ownership bit zero. These flows are considered to belong to *evicted virtual counters*. Also, if a packet assigned to counter index $\leq M$ arrives and finds its corresponding virtual counter with ownership bit one, it sets the counter to one and the ownership bit to zero. This means that the previous virtual counter (of index $> M$) that occupies the same physical position is evicted from the virtual sketch. Note that counters are evicted uniformly at random (because the hash function assigns flows to counters randomly); this is equivalent to randomly discarding flows, also called *flow thinning*. In the example of Figure 1 virtual counter $2M$ is evicted from the virtual sketch. In what follows we assume that evicted counters are discarded.

Flow sampling (see [3]) in its simplest form can be seen as a particular case of the above sketch where number of virtual sketches, Z, goes to infinity, ownership bits are unique flow IDs, and virtual counters can only evict zero-valued counters in the physical sketch. Note that when Z goes to infinity there are no flow collisions in the virtual sketch. However, using a simple flow multiplexing argument, one can show that the amount of flow thinning increases with Z. Thus we want to keep Z as small as possible provided that flow size histogram estimates are accurate. We return to this discussion when evaluating our approach in Section 4.

3.1.2 Sketch histogram

A histogram of the counter values of the virtual sketch is kept in vector $g = (g_0, \ldots, g_{2^b-1})$. This vector is initialized with zero except for $g_0 = 2\,M$. Whenever a counter with value j has its ownership bit changed from one to zero, g_j is decremented by one. This simple

operation reflects the reduction in the number of virtual counters due to contention. The remaining histogram updates are quite trivial.

3.1.3 Pseudo-random auxiliary counters

Our sketch counters perform random (Bernoulli) sampling with probabilities taken from $\{2^{-j} \mid j = 1, \ldots, 2^b - k\}$ for counter values larger than k. In a high level our approach follows the same simple principle of Morris [7], which requires us to perform pseudo-random sampling at line speed. Since traditional pseudo-random number generators are computationally intensive, we instead propose an alternative that is best-case deterministic and worst-case probabilistic. We assume there are N i.i.d. (independent and identically distributed) flows that increment their respective hash counters with probability $1/h$. We start by creating an auxiliary counter c_h and initialize it with $c_h \leftarrow h - 1$. Upon a packet arrival (from any of these N flows) c_h is decremented by one. If $c_h = -1$ we sample the packet (i.e. increment the respective sketch counter) and reinitialize $c_h \leftarrow h - 1$. Note that for $N = 1$ this corresponds to deterministic sampling. Since we only need to maintain one additional counter *per* value h we need $O(\log W)$ auxiliary counters for our sketch. Appendix A shows that as $N \to \infty$ packets are sampled randomly (according to a Bernoulli process) at rate $1/h$, as if we were using a true random number generator.

In the next section we see how the algorithm estimates the flow size histogram from the sketch histogram described above.

3.2 Flow size estimator

In this section we present a flow size estimator that uses the empirical sketch histogram g and outputs a flow size histogram in $O(k^3 + \log W)$ operations. Let the sketch *load* define the number of measured flows divided by the virtual sketch size. Our estimator works as follows: As soon as either the measurement epoch is reached or the load achieves $L = 1/2$, we save g (which is always up-to-date), reinitialize all variables and start another measurement epoch. We use g to refer to the "saved g". With g we can estimate the size histogram using a two step estimator. Section 3.2.1 presents the first step where we estimate flows of size smaller than k (k is the deterministic counting threshold defined in Section 2); Section 3.2.2 presents the second step where we estimate the histogram bins for flow sizes $\geq k$. In this section we also show that for our estimator there is no gain in having sketch counters being able to count more than i packets if we only seek to estimate flows of size smaller than i.

We start with some definitions that are common to Sections 3.2.1 and 3.2.2. Let $\theta = [\theta_i]$, $i = 1, 2, \ldots$ denote the flow size distribution. Note that $\theta_i L$ is the average number of flows of size i associated to a counter at the end of the measurement epoch. In this work we assume that the total number of flows of size $i = 1, 2, \ldots$ measured in an epoch is Poisson distributed. This assumption holds true for our traces and has been reported true for other Internet traces [1]. This is also a fairly weak assumption.

As the total number of flows of size i is Poisson distributed and the hash function randomly assign flows to counters, it is easy to see that the number of flows of size i hashed to a counter are i.i.d. Poisson random variables with parameter $\theta_i L$. Let G be the sum of all elements in g, i.e. $G = \sum_{\forall j} g_j$.

3.2.1 Estimates of flows with size $< k$

Using the above we have the following rather trivial set of equations that describe the relationship between the flow size distribution θ, the counter value histogram g, and the counter load L: The average number of counters with value zero is

$$E[g_0] = G \exp(-L \sum_{i=1}^{\infty} \theta_i) = G \exp(-L), \qquad (1)$$

where $E[g_0]$ is the expected value of g_0. The next equation derives the average number of counters with value one

$$E[g_1] = \theta_1 L E[g_0]. \qquad (2)$$

More generally, for $2 \leq j < k$, we have

$$E[g_j] = \left(\theta_j L + \sum_{m=2}^{j} L^m h_\theta(j, m) \right) E[g_0], \qquad (3)$$

where function h_θ gives the probability that, in a sketch with load one and flow size distribution θ, m flows are hashed into the same counter and their sizes sum up to j; a recursive $O(j^3)$ time algorithm to compute h is given in Appendix C[1].

The sketch load is estimated from equation (1), i.e. $\hat{L} = -\ln(g_0/G)$. Now θ_1 can be easily estimated from equations (1) and (2) as

$$\hat{\theta}_1 = (g_1/g_0)/\hat{L},$$

and the number of flows of size one is estimated as $G\,\hat{\theta}_1$; with $\hat{\theta}_1$ and g_2 we can also estimate θ_2. More generally, we can estimate θ_j using equation (3), $\hat{L}$, and $\hat{\theta}_i$ for $i = 1, \ldots, j-1$

$$\hat{\theta}_j = (g_j/g_0)/\hat{L} - \sum_{m=2}^{j} \hat{L}^{m-1} h_{\hat{\theta}}(j, m). \qquad (4)$$

Thus, estimating all flow sizes with size less than k takes $O(k^3)$ operations if intermediate results are saved. These estimates are quite precise as we will observe in

[1] A simplified version of the non-recursive algorithm is in [5]

the next section. In what follows we see how small sketch loads can help us design a fast estimator for flows of size $\geq k$.

3.2.2 Estimates of larger flows sizes ($\geq k$)

Estimating larger flow sizes encounters a problem: sketch counters are counted probabilistically for values $\geq k$. We can derive an equation similar to (3) that accounts for the probabilistic nature of g_j for $j \geq k$

$$E[g_j] = E[g_0]L \sum_{i=j}^{\infty} \Big(f(i-k, j-k)\,\theta_i + f(i-k, j-k) \sum_{m=2}^{i} L^{m-1}\, h_\theta(i,m) \Big), \quad (5)$$

where function f, described in Appendix B, is the probability that $i-k$ packets triggers $j-k$ increments on a counter with value k. From equation (5) we see that estimating θ_j is not an easy task. In what follows we derive a rough approximation to equation (5) that lead to a very simple estimator. Let

$$B_j = \{k+2^{j-1}-1, \ldots, k+2^j-2\},\ j = k, \ldots, (2^b-1-k) \quad (6)$$

be the bins of our histogram for $j \geq k$ and let

$$\Theta_j = \sum_{\forall i \in B_j} \theta_i, \quad j = k, \ldots, (2^b - 1 - k)$$

be the fraction of all flows with size $i \in B_j$. Assume that j is large. In what follows we approximate the probabilistic counting by deterministic counting, i.e. $f(i-k, j-k) = 1$ if $i \in B_j$ and zero otherwise. We also assume that most of the flow size distribution probability rests in flows with sizes much smaller than 2^j. In this case, collisions of flows whose flow size sums are in B_j are either: (1) a large number of small flow collisions; or (2) few collisions between small and large flows. Let's look at the first case, a large number of small flow collisions. As observed in [5], the probability of 3 or more flow hashing into the same counter is small when $L < 1$. In our case this is particularly true as $L = 1/2$. Thus the effect of a large number of flow collisions is negligible as the summation over h_θ in equation (5) becomes vanishingly small as m increases. Now we consider the second case, a small number of collisions between small and large flows. As j is large, most flows of sizes $i \in B_j$ are at least twice as large as flows of smaller sizes not in B_j. This means that two or three collisions of flows with sizes not in B_j are unlikely to sum up to a size in B_j. Then, apart from degenerated cases like $\sum_{\forall i \in B_j} \theta_i \ll \sum_{\forall i \in B_{j-1}} \theta_i$, most of the collisions between small and large flows that fall into B_j are between small flows and flows whose sizes are in B_j. This motivates us to propose the following approximation to equation (5)

$$E[g_j] \approx E[g_0] L \Theta_j\,.$$

With the above equation we have the following estimate for Θ_j:

$$\hat{\Theta}_j \approx (g_j/g_0)/L\,. \quad (7)$$

Our experiments show that equation (7) is a good approximation to Θ_j (specially for large values of j).

In what follows we evaluate our approach against Internet traces and a synthetic hard-to-estimate distribution.

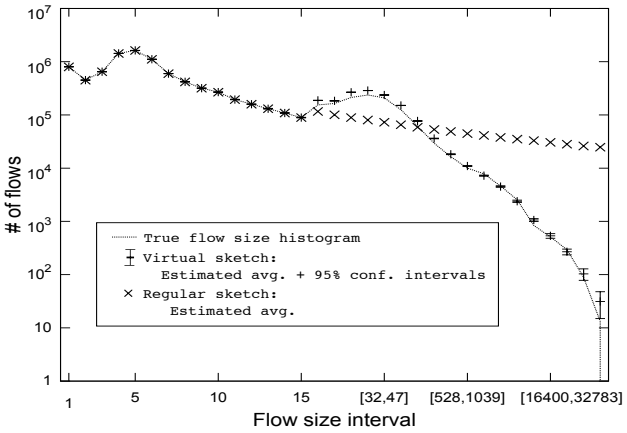

Figure 2: Histogram estimates with 8MB of memory. BB-East-2 trace histogram (line) v.s. histogram estimates (with a virtual sketch and with a regular sketch). Experiment: 9.6 million flows (average), 6 bit counters (7 bits *per* flow), 37 runs.

4. EVALUATION

In this section we evaluate our approach using Internet traces and one synthetic extreme-case distribution. All experiments use parameters: $k = 16$ and $W \approx 10^{14}$ (thus the sketch counter requires $b = 6$ bits). In our first experiment we use the flow size histogram of a Tier 1 backbone trace BB-East-2 described in [8]. This trace has 9.5 million distinct flows collected over a two hour period. This means that an 8MB physical sketch has a 2-fold virtual sketch load $L \approx 1/2$. In our experiments we choose $Z = 2$ (a 2-fold virtual sketch) as it has a low virtual sketch load, L, and $Z = 2$ is the folding value with the smallest flow thinning probability. We use the empirical histogram as a base for generating a series of 37 synthetic traces that will feed our data stream algorithm. In another experiment we repeat the same scenario but replace the multiplexed sketch by a regular sketch. A regular sketch does not need to keep an extra ownership bit and can use this space to reduce its load. Measuring the same number of flows the regular

sketch has load $L = b/(b+1) = 0.86$, in contrast to the load $L = 1/2$ of a 2-fold virtual sketch. Our estimator takes less than one second to compute all estimates in both scenarios. Figure 2 show the results of both experiments. The first experiment (with the virtual sketch) also shows the 95% percentile confidence intervals. For the virtual sketch we can see that our algorithm was able to obtain very good histogram estimates as well asvery tight confidence intervals. Note that for all flow sizes $< k$ our estimator is unbiased. In the case where flow sizes are $\geq k$ we see that equation (7) provides us with estimates that are fairly close to the actual histogram values. On the other hand, the results of our second experiment (with a regular sketch) indicate that a regular sketch performs poorly for flow sizes greater than k.

A more heavily utilized backbone link BB-East-1 from [8] contains 250 million measured flows during a 30 minute interval. This trace requires a 220MB sketch and our estimator still takes less than one second for all estimates. The estimates are even more precise than the ones obtained for BB-East-2, due to the order of magnitude increase in the number of samples.

Next we test our estimator with a histogram whose tail decreases exponentially, i.e. $\theta_i \propto \exp(-\alpha i)$. This is a good extreme-case test of our ability to measure the histogram tail. In this scenario $h_\theta(i, m) = h_\theta(i, m')$ for any $m,\ m' = 1, \ldots, i$, which makes equation (7) a much worst approximation than the case where histograms are heavy tailed. Figure 3 shows the results of an experiment where $\theta_i \propto \exp(-0.01i)$. We can see that our estimator performs reasonably well for flows of size as large as 32,000 packets. Note that we are also able to capture the overall trend of the tail, although the actual values for very large flow sizes are quite over-estimated. However, this flow size histogram is not based on Internet traces and is presented here to assess the performance of our estimator in a worst-case scenario.

5. CONCLUSIONS

In this work we presented a resource minimalist flow size histogram estimator. By trading-off flow size granularity, we count flow sizes in an aggregated and probabilistic manner. This leads to a small memory footprint and an exponential speed up of the time required to compute estimates over previous streaming methods. We tested our scheme using both Tier-1 backbone traces and synthetic distributions with satisfactory accuracy.

6. ACKNOWLEDGMENTS

We thank Jun Xu for many helpful discussions. This work has been supported in part by the National Science Foundation under award number ANI-0325868; and by Sprint ATL and CAPES under award number 2165031.

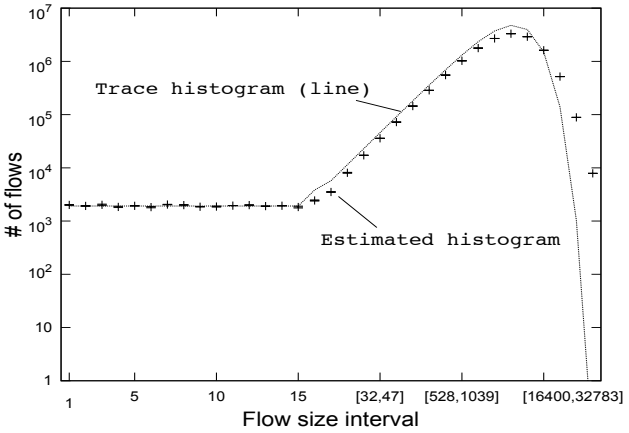

Figure 3: Histogram estimates with 16MB of memory. Exponential histogram (line) with $\theta_i = \exp(-i/10^4)/9999.5$ v.s. histogram estimates (95% confidence interval is too small). Experiment: 20 million flows (average), 6 bit counters (7 bits *per* flow), 43 runs.

Part of this work was carried out when Bruno Ribeiro was an intern at Sprint ATL. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

APPENDIX

A. PSEUDO-RANDOM COUNTING

Here we prove that when N, the number of flows using pseudo-random counter c_h, approaches infinity then packets of any of these flows are sampled according to a Bernoulli process. Initialize $c_h \leftarrow h-1$, remembering that c_h counts down to zero. When c_h is decremented, p is the probability that a given flow decremented it and $1-p$ be the probability that other flows decremented it. Let X be the number of subtractions by a flow between c_h rollovers (a rollover is when counter goes instantly from $c_h = 0$ back to $c_h = h$). Define

$$P_m = P[X = m], m = 1, \ldots, h-1.$$

It is easy to see that if we assume that $\binom{k}{j} = 0$ whenever $j > k$ we have

$$P_m = \binom{h-1}{m-1}(1-p)^{h-m}p^m \quad + \\ \sum_{i=0}^{m-1}\binom{h-1}{i}(1-p)^{h-i}\,p^i\,P_{m-i},\ m = 1, 2, \ldots, h. \tag{8}$$

Now we prove that the flow is Bernoulli sampled. Proof

by induction on m.

$$P_1 = (1-p)^{h-1}p + (1-p)^h P_1 \rightarrow P_1 = \frac{(1-p)^{h-1}p}{1-(1-p)^h}$$

as $N \rightarrow \infty$, $p \rightarrow 0$ and then $P_1 = 1/h$.

Induction:
Assume $P_i = (1-1/h)^{i-1}1/h$ for $i = 1, 2, \ldots, m-1$. Then equation (8) becomes

$$\begin{aligned} P_m =& (h-1)p\,P_{m-1} + (1-p)^h P_m + O(p^2) \\ =& (h-1)p(1-1/h)^{m-2}1/n + (1-p)^h P_m + O(p^2), \end{aligned}$$

passing all variables P_m to the left side

$$\begin{aligned} P_m =& ((h-1)p(1-1/h)^{m-2}1/n + O(p^2))/(1-(1-p)^h) \\ =& (1-1/h)^{m-1}1/h + O(p). \end{aligned}$$

As $p \rightarrow 0$, $P_m = (1-1/h)^{m-1}1/h$, which is geometrically distributed and thus the flow is Bernoulli sampled.

B. COUNTER INCREMENT PROBABILITY

The probability of having j counter increments out of i packets is given by $f(i,j) = 2^{-(j(j+1)/2)} f'(i-j, j+1)$, where

$$f'(i,j) = \begin{cases} \sum_{m=0}^{i} (1-2^{-j})^m f'(i-m, j-1) & \text{if } j \geq 2 \\ (1-2^{-1})^i & \text{otherwise.} \end{cases}$$

C. FLOW COLLISION FUNCTION

Function $h_\theta(j,m) = h'_\theta(j,1,m)$ can be computed using the following recursion.

$$h'_\theta(j,w,m) = \begin{cases} \exp(-\sum_i \theta_i) & \text{if } j = m = 0 \\ 0 & \text{if } w > j \text{ or } w\,m > j \end{cases}$$

otherwise

$$h'_\theta(j,w,m) = \sum_{r=0}^{\min(\lfloor j/w \rfloor, m)} \frac{(\theta_w)^r}{r!}\, h'_\theta(j - r\,w, w+1, m-r);$$

caching its intermediate results h_θ is known to have complexity $O(j^3)$ [5].

1. REFERENCES

[1] Chadi Barakat, Patrick Thiran, Gianluca Iannaccone, Christophe Diot, and Philippe Owezarski. Modeling internet backbone traffic at the flow level. *IEEE Transactions on Signal Processing*, 51(8):2111–2124, August 2003.

[2] Graham Cormode, S. Muthukrishnan, and Irina Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *VLDB*, pages 25–36, 2005.

[3] Nick Duffield, Carsten Lund, and Mikkel Thorup. Estimating flow distributions from sampled flow statistics. *IEEE/ACM Transactions on Networking*, 13(5):933–946, 2005.

[4] Abhishek Kumar, Minho Sung, Jun Xu, and Ellen W. Zegura. A data streaming algorithm for estimating subpopulation flow size distribution. In *Proceeding of the ACM SIGMETRICS*, pages 61–72, 2005.

[5] Abhishek Kumar, Minho Sung, Jun (Jim) Xu, and Jia Wang. Data streaming algorithms for efficient and accurate estimation of flow size distribution. In *Proceeding of the ACM SIGMETRICS*, pages 177–188, 2004.

[6] Yi Lu, Andrea Montanari, Balaji Prabhakar, Sarang Dharmapurikar, and Abdul Kabbani. Counter braids: A novel counter architecture for per-flow measurement. In *Proceeding of the ACM SIGMETRICS*, 2008.

[7] Robert Morris. Counting large numbers of events in small registers. *Communications of the ACM*, 21(10):840–842, 1978.

[8] Bruno Ribeiro, Don Towsley, Tao Ye, and Jean Bolot. Fisher information of sampled packets: an application to flow size estimation. In *ACM Internet measurement conference*, pages 15–26, 2006.