2015 IEEE 23rd International Conference on Network Protocols

Counter Tree: A Scalable Counter Architecture for Per-Flow Traffic Measurement

Min Chen Shigang Chen
Department of Computer & Information Science & Engineering
University of Florida, Gainesville, FL 32611, USA
Email:{min, sgchen}@cise.ufl.edu

***Abstract*—Per-flow traffic measurement, which is to count the number of packets for each active flow during a certain measurement period, has many applications in usage accounting, traffic engineering, service provision and anomaly detection. In order to maintain the high throughput of routers or switchers, the per-flow traffic measurement module should use high-bandwidth SRAM that allows fast memory accesses. Due to the limited SRAM space, exact counting, which requires to keep a counter for each flow, does not scale to large networks consisting of numerous flows. Some recent work takes a different path to accurately estimate the flow sizes using counter architectures that can fit into tight SRAM. However, existing counter architectures have some limitations, either still requiring considerable SRAM space, or having a very small estimation range. In this paper, we design a scalable counter architecture Counter Tree which leverages a two-dimensional counter sharing scheme to achieve far better memory efficiency and significantly extend estimation range. The extensive experiments with real network trace demonstrate that our counter architecture can produce accurate estimates for flows of all sizes even under a very tight memory space, e.g., 2 bits per flow.**

I. Introduction

Per-flow traffic measurement is one of the fundamental problems in network traffic measurement [1]–[9]. In a general definition, it is to count the number of packets (or called flow size) for each active *flow* during a measurement period. The flows under measurement can be per-source flows, per-destination flows, per-source/destination flows, TCP flows, WWW flows, P2P flows, or any user-defined logical flows. Each flow is uniquely identified by its *flow label*, e.g., the flow labels for per-source flows are the source addresses. Per-flow traffic measurement has many important applications in usage accounting, traffic engineering, service provision and anomaly detection of large networks [10]–[13]. For example, ISPs can use the per-flow information to optimize traffic routing in backbones to reduce congestion; per-flow measurement can also help determine the types of traffic transmitted in the networks, sent from a particular source, or destined to a particular address; network administrators can perform per-source traffic measurement to estimate the scanning rates of worm-infected hosts in a worm attack.

We stress that per-flow traffic measurement significantly differs from a related problem called *flow cardinality estimation* [14]–[18], which is to estimate the number of *distinct* elements in each flow. Consider a per-source flow. Suppose the source sends 1,000 packets to a single destination during a measurement period. The flow size is 1,000 in terms of the number of packets, but the flow cardinality is 1 if destination addresses are considered as elements. The related work of flow cardinality estimation will be introduced in Section VIII.

Challenge and Prior Art: The challenge of per-flow traffic measurement mainly results from the lack of affordable high-density high-bandwidth memory devices. Commercially available DRAM, whose access time is in the order of tens of nanoseconds, cannot keep up with the dramatically increasing line speed (NEC and Corning achieves a transmission rate of 1.05 Petabit/s using a 12-core fiber design [19]). On the contrary, SRAM with much smaller access time has very low density. As a result, large SRAM is expensive and difficult to implement on-chip. Moreover, the already limited SRAM is shared among different functions, such as routing, scheduling, traffic measurement, and security. Even for traffic measurement alone, there can be multiple functions performed concurrently, each requiring some SRAM. Therefore, the SRAM dedicated for per-flow traffic measurement can be extremely small, necessitating memory-efficient measurement approaches. In addition, we need to minimize the processing time of per-flow traffic measurement, particularly the number of memory accesses, such that the implementation of the measurement module will not deteriorate throughput of routers or switchers.

With tremendous number of flows in the networks, it is impossible to keep a counter for each flow in SRAM. *Exact counting* generally adopts a hybrid SRAM-DRAM architecture [1]–[3], where small counters in SRAM are incremented at high speed, and occasionally written back to larger counters in DRAM. However, the hybrid architecture incurs very costly SRAM-to-DRAM updates. Furthermore, the flow-to-counter association requires considerable SRAM (at least 10MB [5]).

To fit the measurement module in tight SRAM, some schemes only give the distribution of flow sizes [20], [21], or measure the sizes of large flows [22], [23]. Some recent work takes a different path to accurately estimate the flow sizes instead of counting their exact sizes, thereby reducing storage overhead. The state-of-art estimation approaches include bitmap-based MSCBF, and counter-based Counter Braids and randomized counter sharing scheme.

The *Multiresolution Space-Code Bloom Filter* (MSCBF) [4] employs multiple Bloom filters [24] to encode packets with different sampling probabilities. Filters with high sampling probabilities can keep track of small flows, while filters with low sampling probabilities can track large flows. However, the bitmap nature of MRSCBF determines that it is not memory-efficient for counting [7].

The *Counter Braids* (CB) [5], [6] is a counter architecture for flow size measurement. It avoids the storage of flow-to-counter association by hashing flows to counters on the fly, and it reduces memory requirement by sharing counters among flows. A typical implementation of CB consists of two layers of counters, and employs three hash functions. To encode a packet, it is hashed to three counters based on its flow label, which are all incremented by one. If any of the first-layer counter overflows, another three second-layer counters will be used. Since each counter is shared by multiple flows, it counts all associated flows. Therefore, the counters essentially form a set of linear equations of the flow sizes. A message passing reconstruction algorithm was proposed to estimate the flow sizes in an iterative way. CB can recover the exact flow sizes when sufficient memory is available, e.g., 10 bits per flow. However, CB has three limitations. First, it performs 6 (occasionally 12) memory accesses to encode one packet. Second, it yields very biased or even meaningless estimates under a tight memory, e.g., less than 4 bits per flow. In fact, we find that the estimation results of CB do not converge even with 8 bits per flow, though it may occasionally produce very accurate results if we manually terminate the process after some iterations. Third, CB does not support instantaneous queries of flow sizes. All flow sizes must be decoded together at the end of a measurement period.

A new data encoding/decoding scheme, called *randomized counter sharing* [7], was proposed to further reduce the memory requirement and processing time of per-flow traffic measurement. The idea is to split each flow among a number of counters (called the *storage vector* of the flow) that are randomly selected from a counter pool. When encoding a packet of a particular flow, it is randomly mapped to a counter of the flow's storage vector, and the counter is then incremented by one. This scheme requires only 2 memory accesses for encoding one packet, achieving the optimal processing speed. Moreover, it can still yield reasonably accurate estimates under a tight memory space where CB no longer works. Two estimation methods CSM and MLM are used to estimate flow sizes. The most serious problem of this scheme is that its estimation range is limited, e.g., a few thousands in a typical implementation. For large flows with sizes beyond the estimation range, the scheme leads to very negatively biased estimates since overflowed counters lose information. In the journal version [9], some approaches were provided to extend the estimation range, which however cannot address the issue fundamentally. The first approach is to increase the length of each counter or the size of the storage vector. However, this approach degrades estimation accuracy since fewer counters are available or each counter is shared by more flows. Following a reasonable parameter setting, the estimation range is still very limited. The second approach employs a sampling module. Each arriving packet is sampled with a probability p before being encoded to a counter. Aggressive sampling not only introduces significant error [4], but also fails to measure some small-size or even moderate-size flows. For example, if we let $p = 0.001$, flows with sizes less than 1,000 are hardly be captured. The final approach resorts to the hybrid SRAM/DRAM design, which requires costly SRAM-to-DRAM updates.

Our Contributions: To address the issues of existing counter architectures, we design Counter Tree, a novel SRAM-only counter architecture. The contributions of this paper are summarized as follows:

1) We propose a two-dimensional counter sharing scheme, where each counter can be shared not only by different flows, but also among different virtual counters. Thanks to this scheme, a significant memory save can be achieved, making Counter Tree work well under a tight memory where CB does not work.
2) Counter Tree reserves more significant bits for larger flows, which dramatically extends the estimation range when compared with the randomized counter sharing scheme.
3) Counter Tree has a very high processing speed. Encoding a packet only requires a little more than 2 memory accesses on average, which is asymptotically optimal.
4) Counter Tree supports an instantaneous query of the size of an arbitrary flow. Two offline decoding methods are proposed to estimate flow sizes. The extensive experiments with real network trace demonstrate both methods can generate accurate results even under extremely tight memory, e.g., 2 bits per flow.

II. Performance Metrics

In this paper, we employ three metrics to evaluate the performance of different per-flow traffic measurement schemes:

Memory requirement: Due to the constraint of SRAM space, we want to use as small memory (in the sequel memory refers to SRAM) as possible to achieve per-flow traffic measurement. Here we focus on the memory requirement for implementing the counter architectures, while the collection of flow labels is beyond the scope of this paper. Some memory-efficient schemes [6] for flow label collection can be found in literature.

Processing time: To keep up with the line speeds, the processing time for encoding a packet should be small, such that the implementation of the measurement module will not deteriorate throughput. In most counter architectures [4], [5], [7], the processing time for encoding a packet mainly results from the memory accesses and hash computations.

Estimation accuracy: Given a particular memory space, the estimates of flow sizes are desirable to be as accurate as possible. Suppose the true size of a flow is s, and the estimated size is $\hat{s}$. We use the relative bias $Bias(\frac{\hat{s}}{s})$ and relative standard error $StdErr(\frac{\hat{s}}{s})$ to evaluate the estimation accuracy,

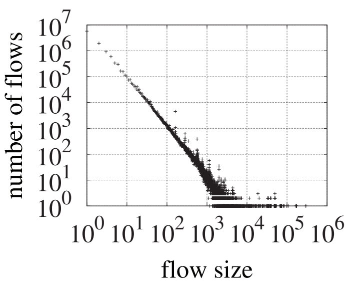

Fig. 1: Distribution of flow sizes, where each point represents the number (y coordinate) of flows that have a particular size (x coordinate).

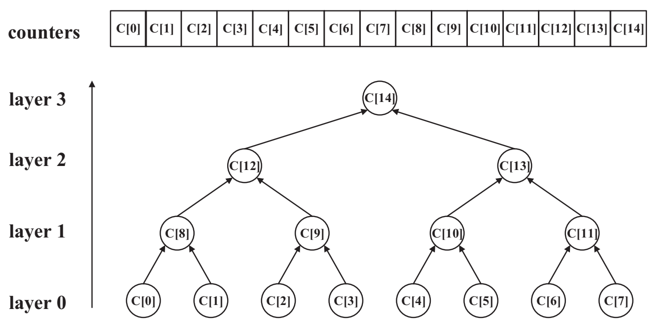
counters
C[0] C[1] C[2] C[3] C[4] C[5] C[6] C[7] C[8] C[9] C[10] C[11] C[12] C[13] C[14]
layer 3
layer 2
layer 1
layer 0

Fig. 2: An example of organizing counters into a binary tree.

which are defined as follows:

$$Bias(\frac{\hat{s}}{s}) = E(\frac{\hat{s}}{s}) - 1, \tag{1}$$

$$StdErr(\frac{\hat{s}}{s}) = \sqrt{Var(\frac{\hat{s}}{s})} = \frac{\sqrt{Var(\hat{s})}}{s}. \tag{2}$$

III. Design of Counter Tree Architecture

A. Motivation

In spite of the large number of flows in networks, many studies reveal a common observation that a small percentage of large flows account for a high percentage of the traffic (also known as the heavy-tailed distribution). The study in [25] showed that 9% of the flows account for 90% of the byte traffic. As an example, we use a network trace obtained from the main gateway of our university, which contains about 68 million TCP flows and 750 million packets. The distribution of flow sizes is illustrated in Fig. 1, where each point represents the number (y coordinate) of flows that have a particular size (x coordinate). This log-scale figure demonstrates that the vast majority of flows have small sizes, while only a small number of flows have large sizes. Without knowing the flow sizes beforehand (which are in fact what we want to measure), the length of counters should be set according to the maximum flow size, which may need to be as large as 64 bits [2]. However, if a flow turns out to be small, e.g., with a size of 1, most of the bits in its counter will be wasted.

B. Two-dimensional Counter Sharing

To reduce the memory waste caused by small flows, we should enable counter sharing. In this paper, we propose a novel counter sharing scheme called two-dimensional counter sharing, including horizontal counter sharing and vertical counter sharing.

For horizontal counter sharing, each counter is shared by multiple flows. The rationale is to let large flows *borrow* memory from small flows that will not fully use their counters. As an example, consider two flows f and g with sizes 2 and 58, respectively. If we keep a counter for each flow, the counter length should be set according to g's size to avoid overflow, i.e., at least 6 bits. Obviously, most bits in f's counter are wasted. Therefore, we want to propose a mechanism to allow g to borrow unused bits from f.

Using horizontal counter sharing allows part of the bits wasted by small flows to be reused by large flows. However, since small flows take a dominant percentage of network flows, many bits in counters occupied by only small flows can still be wasted. This observation inspires the idea of vertical counter sharing, which makes the more significant bits (higher-order bits) be shared by multiple counters. To do so, we introduce the concept of *virtual counter*. Each virtual counter is comprised of multiple small counters, where a counter representing more significant bits is shared by more other virtual counters. Vertical counter sharing in nature is equal to dynamic memory allocation based on flow sizes (which is however more difficult since we do not know the flow sizes beforehand). The more significant bits are reserved for large flows that can use them on demand. Following the previous example, suppose we have three counters, each with 3 bits. The first counter is allocated to f, the second is for g, while the third is reserved for whoever needs it. As a result, g will use the third counter when the second counter overflows. The three counters only require 9 bits to successfully encode f and g.

The scheme of two-dimensional counter sharing contributes to significant memory save, but it also introduces noise among counters. The good thing is that we can employ some statistical tools to remove such noise as we will show shortly.

C. Counter Tree Architecture

We design a Counter Tree architecture to achieve two-dimensional counter sharing. Given a memory space of M bytes, we divide it into small counters, each consisting of b bits. We organize those counters into a tree structure from the bottom up. The degree of each non-leaf node is d. Let the leaf nodes be the layer 0, and M' bytes be allocated for layer-0 counters, which translates into $\frac{M'}{b}$ counters. Hence, the height of the counter tree can be up to $\log_d \frac{M'}{b} + 1$. Since the number of counters on the jth layers is reduced to $\frac{1}{d}$ of the $(j-1)$th layer, the following memory constraint should hold

$$\sum_{j=0} \frac{M'}{d^j} \leq M. \tag{3}$$

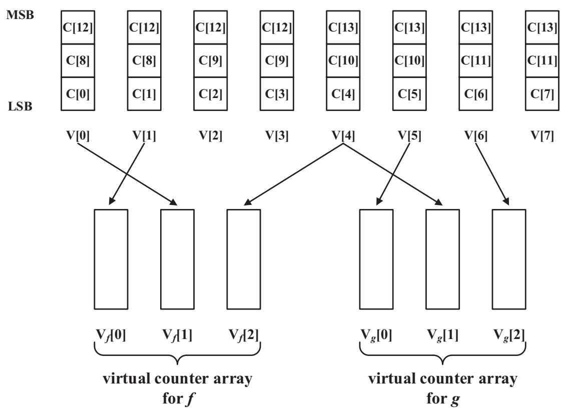

Fig. 3: Virtual counters and virtual counter arrays for flows. The virtual counter array for a particular flow consists of multiple counters pseudorandomly chosen from the counters.

Since $\sum_{j=0} \frac{M'}{d^j} < \frac{d}{d-1}M'$, we let $\frac{d}{d-1}M' = M$. Hence,

$$M' = \frac{d-1}{d}M, \tag{4}$$

and the number m of counters on layer 0 is

$$m = \frac{M'}{b} = \frac{(d-1)M}{bd}. \tag{5}$$

It means that only $\frac{1}{d}$ of the memory will be reserved for non-leaf counters. Fig. 2 gives an example of organizing the 15 counters into a binary tree with 4 layers, where $m = 8$ and $d = 2$. Note that some high-layer counters may not be used in practice. Hence, we define the *effective height* h of the Counter Tree as the number of layers that have at least one used counters (the counter value is not zero). The value of h increases when more packets are encoded. Starting from a leaf node $C[i]$ ($0 \le i < m$), the h counters along the path to the root form a virtual counter, denoted by $V[i]$. As a result, there will be m virtual counters in total, denoted by V. Starting from $C[i]$ at layer 0, the counter at layer j ($0 \le j < h$) that $V[i]$ will include, denoted by $V[i][j]$, is

$$V[i][j] = C[\lfloor \frac{i}{d^j} \rfloor + \sum_{t=1}^{j} \frac{m}{d^{t-1}}]. \tag{6}$$

Suppose $C[14]$ has not been used and therefore $h = 3$. The Counter Tree in Fig. 2 can yield 8 virtual counters as shown in the upper half of Fig. 3

D. Counting Range

In Counter Tree, each virtual counter can have up to $b \times (\log_d \frac{M'}{b} + 1)$ bits. As a result, the counting range of each virtual counter can be as large as $2^{b(\log_d \frac{M'}{b}+1)}$ (in number of packets). In contrast, the counting range of each counter is only $2^b - 1$. Let us use specific numbers to demonstrate how tremendous the improvement is. Suppose $M = 1\text{MB}$, $b = 4$, $d = 2$, and M' is therefore 0.5MB according to 4. Hence, the counting range of each counter is $2^4 - 1 = 15$, while each virtual counter can count up to $2^{4(\log_2 \frac{0.5\text{MB}}{4\text{bit}}+1)} = 2^{84}$ packets. Therefore, Counter Tree can scale to measure extremely large flows.

E. Design Overview

Our traffic measurement function using Counter Tree consists of two modules. The online data encoding module stores the information of arriving packets in the Counter Tree. For each packet, it is mapped to a virtual counter by one hash computation and then the virtual counter will be updated, which needs approximately two memory accesses. At the end of each measurement period, the Counter Tree is stored to the disk and all counters are then reset to zeros. The offline data decoding module estimates the flow sizes. It is performed by a designated offline computer. We propose two methods for separating the information about the size of a flow from the noise in the virtual counters. The first one is called Counter Tree base Estimation (CTE). The second one is based on the maximum likelihood estimation method (CTM). Both methods can yield accurate estimates for flow sizes.

IV. Online Packet Encoding

In this section, we show how to encode a packet to the Counter Tree.

A. Encoding

Consider an arbitrary flow f. We pseudorandomly choose r out of the m virtual counters to logically form a *virtual counter array* of f, denoted by V_f. The selection can be achieved by applying r independent hash functions to the flow label. Hence, the ith counter of V_f, denoted by $V_f[i]$, is chosen from V as follows

$$V_f[i] = V[h_i(f)], \tag{7}$$

where $0 \le i < r$ and $h_i(\cdot)$ is a hash function $\in [0, m-1]$. To reduce the overhead of implementing r independent hash functions, we can use one master hash function H and a set S of random seeds, and let

$$h_i(f) = H(f \oplus S[i]), \tag{8}$$

where $\oplus$ is the XOR operator. The bottom half of Fig. 3 illustrates the virtual counter arrays for f and g, where $r = 3$ and the virtual counter $V[4]$ is shared by both flows.

At the beginning of each measurement period, all counters are initialized to 0s. When a packet of flow f arrives, the router extracts its flow label f, randomly chooses a virtual counter from V_f, and increments that virtual counter by 1. More specifically, the router generates a random number $i \in [0, r-1]$, computes the hash value $u = h_i(f)$, and sets $V[u] = V[u]+1$. Note that the update of $V[u]$ may involve the updates of multiple counters. According to (6), the router first fetches counter $C[u]$ from memory and increases it by 1. If $C[u]$ does not overflow, the encoding for this packet is done. Otherwise, the reader further needs to fetch $C[\lfloor \frac{u}{d} \rfloor + m]$, and add the overflowed 1 to $C[\lfloor \frac{u}{d} \rfloor + m]$. The process continues until no overflow happens, or the counter on the root has been used. Note that if the counter on the root still overflows

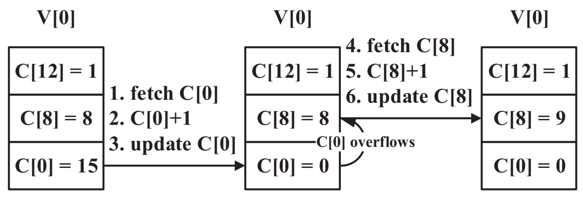

Fig. 4: The process for encoding a packet to a virtual counter.

(though it rarely occurs in our design), its updated value should not be written back. Fig. 4 gives an example of the online encoding process for a packet of f. Suppose $b = 4$ (i.e., the counting range of each counter is 15), currently $h = 3$, and $V[0]$ is chosen for encoding that packet. The router first fetches $C[0]$ whose current value is 15. After adding 1 to $C[0]$, $C[0]$ becomes 0 and leads to an overflow. Hence, the router writes back $C[0] = 0$, further fetches $C[8]$ with value 9, and calculate $C[8] = C[8] + 1$. Since $C[8] = 10$ does overflow, the router just writes it back and the encoding process terminates.

B. Number of Memory Accesses

To encode a packet, the router at least needs to read and write 1 counter, which requires 2 memory accesses. Hence, the lower bound of the number of memory accesses for encoding a packet is 2. In the worst case, the router needs to update h counters, which requires $2h$ memory accesses. The good thing is that the router needs to fetch another counter only when the current counter overflows. Hence, we have the following theorem:

Theorem 1. *The upper bound of the amortized number of memory accesses for encoding a packet to Counter Tree is* $2 + \frac{2}{2^b-1}$*, where* b *is length of each counter.*

Proof: Consider the extreme case that all flows share the same virtual counter. This leads to the most overflows since all packets are encoded to same counters. Because each counter consists of b bits, the jth layer counter of the virtual counter overflows after encoding every $\frac{1}{2^{(j+1)b}}$ packets. It takes 2 memory accesses to read and write a counter. Therefore, the amortized number of memory accesses for encoding a packet is

$$2 \times \sum_{j=0}^{h-1} \frac{1}{2^{jb}} = \frac{2(1 - \frac{1}{2^{bh}})}{1 - \frac{1}{2^b}} < 2 + \frac{2}{2^b - 1}. \tag{9}$$

In Counter Tree, packets are randomly mapped to different virtual counters, which significantly reduces the number overflows. Therefore, $2 + \frac{2}{2^b-1}$ gives an upper bound of the amortized number of memory accesses for encoding a packet. ■

Fig. 5 shows the upper bound of the amortized number of memory accesses for encoding a packet with respect to b. We find that it quickly converges to 2, the lower bound, with the increase of b.

V. Counter Tree based Estimation

After the measurement period, offline estimation can be performed to recover flow sizes from the Counter Tree. In this section, we propose and analyze the Counter Tree based Estimation (CTE) method. To estimate the size of an arbitrary flow f, we first add up the values of the virtual counters f is mapped to, and then subtract the noise introduced by other flows from the sum. Thanks to the random mapping, the expected value of such noise can be calculated in a probabilistic way.

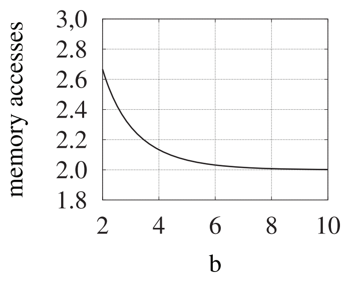

Fig. 5: Amortized number of memory accesses with b.

A. CTE method

Consider the ith virtual counter $V_f[i]$ in the virtual counter array of flow f. According to (7), $V_f[i] = V[u]$, where $u = h_i(f)$. We know $V[u]$ encodes part of f's packets, as well as the noise introduced by other flows. There are two sources of noise: First, $V[u]$ is shared by other flows; Second, all component counters in $V[u]$ except $C[u]$ are shared by other virtual counters. To accurately recover the number of f's packets encoded by $V[u]$, we need to figure out how to remove such noise.

With a effective height h, the component counter of $V[u]$ at the highest layer is $C[v]$, where $v = \lfloor \frac{u}{d^{h-1}} \rfloor + \sum_{t=1}^{h-1} \frac{m}{d^{t-1}}$ according to (6). Consider the subtree rooted at $C[v]$, denoted by T. T consists of d^{h-1} leaf nodes, which correspond to d^{h-1} virtual counters. Let $k = d^{h-1}$. Due to counter sharing, any flows mapped to those k virtual counters may finally introduce noise to $V[u]$. Therefore, we treat T as a whole, called a it subtree counter, when dealing with noise. We denote the value of T by a random variable X_i. As an example, in Fig. 2 the value of the subtree counter rooted at $C[12]$ is $C[12] \times 2^{2b} + (C[8] + C[9]) \times 2^b + (C[0] + C[1] + C[2] + C[3])$. Let random variable Y_i be the portion of X_i contributed by flow f, and Z_i be the portion of X_i contributed by all other flows. Hence, $X_i = Y_i + Z_i$.

Suppose the true flow size of f during the measurement period is s, and the sum of all flow sizes is n. Each packet of f has a probability $\frac{1}{r}$ to be mapped to $V[u]$ and increment it by one. Therefore, we have the following lemma:

Lemma 1. *Y_i follows a binomial distribution:*$Y_i \sim B(s, \frac{1}{r})$.

For the distribution of Z_i, we have the following lemma:

Lemma 2. *Z_i follows a binomial distribution* $Z_i \sim B(n, \frac{k}{m})$.

Proof: Consider an arbitrary flow g. For a particular virtual counter in T, it has a probability $1 - \frac{\binom{m-1}{r}}{\binom{m}{r}} = \frac{r}{m}$

to be chosen by g. Therefore, one of the k virtual counters in T is chosen by g is $\binom{k}{1} \times \frac{r}{m} = \frac{kr}{m}$. Hence, the probability that a packet of flow g is encoded by any virtual counter in T is $\frac{kr}{m} \times \frac{1}{r} = \frac{k}{m}$. Therefore, $Z_i \sim B(n-s, \frac{k}{m})$. Since $n \gg s$, we have $Z_i \sim B(n, \frac{k}{m})$. ■

Given the distributions of Y_i and Z_i, we know $E(Y_i) = \frac{s}{r}$, and $E(Z_i) = \frac{nk}{m}$. Therefore, we have

$$E(X_i) = E(Y_i + Z_i) = E(Y_i) + E(Z_i) = \frac{s}{r} + \frac{nk}{m}.$$

Hence, we obtain

$$s = rE(X_i) - \frac{nkr}{m}. \tag{10}$$

Since the virtual counter array of f consists of r virtual counters, which correspond to r subtree counters. The values of those r subtree counters can roughly be treated as independent and identically distributed random variables. We can replace $E(X_i)$ with $\frac{\sum_{i=0}^{r-1} X_i}{r}$, and obtain an estimator for s as follows

$$\hat{s} = \sum_{i=0}^{r-1} X_i - \frac{nkr}{m}. \tag{11}$$

B. Analysis of $\hat{s}$

We first provide and prove the following theorem:

Theorem 2. *$\hat{s}$ is an unbiased estimator for s.*

Proof: We can calculate

$$E(\hat{s}) = E(\sum_{i=0}^{r-1} X_i) - \frac{ndr}{m} = r(\frac{s}{r} + \frac{nk}{m}) - \frac{nkr}{m} = s. \tag{12}$$

Therefore,$\hat{s}$ is an unbiased estimator for s. ■

Next, we analyze the variance of $\hat{s}$. Since Y_i and Z_i follow binomial distributions, we have

$$Var(Y_i) = \frac{s}{r}(1 - \frac{1}{r}), \quad Var(Z_i) = \frac{nk}{m}(1 - \frac{k}{m}). \tag{13}$$

In addition, Y_i and Z_i are independent with each other. Hence, $Cov(Y_i, Z_i) = 0$. Hence, we have

$$\begin{aligned} Var(X_i) &= Var(Y_i) + Var(Z_i) + 2Cov(Y_i, Z_i) \\ &= \frac{s}{r}(1 - \frac{1}{r}) + \frac{nk}{m}(1 - \frac{k}{m}). \end{aligned} \tag{14}$$

Hence, we have

$$\begin{aligned} Var(\hat{s}) &= Var(\sum_{i=0}^{r-1} X_i) = r^2 Var(X_i) \\ &= s(r-1) + \frac{nkr^2}{m}(1 - \frac{k}{m}) \\ &= s(r-1) + \frac{nr^2bd^h}{(d-1)M}(1 - \frac{bd^h}{(d-1)M}), \end{aligned} \tag{15}$$

where we have used $m = \frac{(d-1)M}{bd}$ and $k = d^{h-1}$. In addition, the relative standard error of the estimator is

$$StdErr(\frac{\hat{s}}{s}) = \frac{\sqrt{s(r-1) + \frac{nr^2bd^h}{(d-1)M}(1 - \frac{bd^h}{(d-1)M})}}{s}. \tag{16}$$

Taking the partial derivatives of $Var(\hat{s})$ with respect to M, r, d, and b, respectively, we have

$$\begin{aligned} \frac{\partial Var(\hat{s})}{\partial M} &= \frac{nr^2bd^h}{M^2}(-1 + \frac{2bd^h}{(d-1)M}), \\ \frac{\partial Var(\hat{s})}{\partial r} &= s + \frac{2nrbd^h}{(d-1)M}(1 - \frac{bd^h}{(d-1)M}), \\ \frac{\partial Var(\hat{s})}{\partial b} &= \frac{nr^2d^h)}{(d-1)M}(1 - \frac{2bd^h}{(d-1)M}), \\ \frac{\partial Var(\hat{s})}{\partial d} &= \frac{nr^2bd^{h-1}}{(d-1)M}(h - \frac{d}{d-1})(1 - \frac{2bd^h}{(d-1)M}). \end{aligned} \tag{17}$$

We know $\frac{bd^h}{d-1}$ is close to the number of bits in each subtree counter, which is generally much smaller than M. Therefore, $\frac{\partial Var(\hat{s})}{\partial M} < 0$, $\frac{\partial Var(\hat{s})}{\partial r} > 0$, $\frac{\partial Var(\hat{s})}{\partial b} > 0$, and $\frac{\partial Var(\hat{s})}{\partial d} > 0$, implying the variance of $\hat{s}$ increases with the increase of r, b or d, but decreases with the increase of M. Since the counting range of each virtual counter array is proportional to $r2^{bh}$, there exists a tradeoff between estimation accuracy and estimation range. Since the counting range of each virtual counter array is proportional to $r2^{bh}$, we suggest a parameter setting such that $r2^{bh}$ is no less than the estimated largest size of all flows.

C. Confidence Interval

When n is large enough, the binomial distribution $Z_i \sim B(n, \frac{k}{m})$ approximates to a normal distribution, namely $Z_i \overset{\text{approx}}{\sim} N(\frac{nk}{m}, \frac{nk}{m}(1 - \frac{k}{m}))$. Similarly, $Y_i \overset{\text{approx}}{\sim} N(\frac{s}{r}, \frac{s}{r}(1 - \frac{1}{r}))$. Since the linear combination of independent random variables that follow normal distributions also follow normal distribution, $X_i \overset{\text{approx}}{\sim} N(\mu, \sigma^2)$, where $\mu = \frac{nk}{m} + \frac{s}{r}$, and $\sigma^2 = \frac{nk}{m}(1 - \frac{k}{m}) + \frac{s}{r}(1 - \frac{1}{r})$. According to (11), we know

$$\hat{s} \overset{\text{approx}}{\sim} N(s, s(r-1) + \frac{nkr^2}{m}(1 - \frac{k}{m})). \tag{18}$$

Therefore, the $1 - \alpha$ confidence interval for s is

$$\hat{s} \pm Z_{\frac{\alpha}{2}} \sqrt{s(r-1) + \frac{nkr^2}{m}(1 - \frac{k}{m})}, \tag{19}$$

where $Z_{\frac{\alpha}{2}}$ is the $1 - \frac{\alpha}{2}$ percentile for the standard normal distribution.

VI. Counter Tree based Maximum Likelihood Estimation

In this section, we provide and analyze another estimator for flow sizes called Counter Tree based Maximum likelihood Estimation (CTM). Our analysis in Section V-A shows that for an arbitrary virtual counter in f's virtual counter array, a packet of f has a higher probability to be mapped to this virtual counter than any packets from other flows. Therefore, we can use maximum likelihood estimation method to estimate the flow size of f, such that the probability for observing the given counter values in f's virtual counter array is maximized.

A. CTM method

According to Lemma 2, the probability for $Z_i = z_i$ is

$$P(Z_i = z_i) = \binom{n}{z_i}(\frac{k}{m})^{z_i}(1-\frac{k}{m})^{n-z_i}.$$

Since the value of n is known, and the values of m and k are determined by prescribed system parameters M, b and d, h, $P(Z_i = z_i)$ can be written as a function of z_i, which is $p(z_i)$. Therefore, the probability for observing $X_i = x_i$ can be calculated by

$$\begin{aligned}P(X_i = x_i) &= P(Y_i + Z_i = x_i) \\ &= \sum_{z_i=0}^{x_i} p(z_i)P(Y_i + Z_i = x_i|Z_i = z_i) \\ &= \sum_{z_i=0}^{x_i} p(z_i)P(Y_i = x_i - z_i) \\ &= \sum_{z_i=0}^{x_i} p(z_i)\binom{s}{y_i}(\frac{1}{r})^{y_i}(1-\frac{1}{r})^{s-y_i} \\ &= \sum_{z_i=0}^{x_i} p(z_i)q(s, y_i),\end{aligned} \tag{20}$$

where $y_i = x_i - z_i$, $q(s, y_i) = \binom{s}{y_i}(\frac{1}{r})^{y_i}(1-\frac{1}{r})^{s-y_i}$, and we have used $Y_i \sim B(s, \frac{1}{r})$. Hence, the likelihood function for observing $X_0 = x_0$, $X_1 = x_1$, ..., $X_{r-1} = x_{r-1}$ is

$$L(s; x_0, x_1, \ldots, x_{r-1}) = \prod_{i=0}^{r-1}\sum_{z_i=0}^{x_i} p(z_i)q(s, y_i). \tag{21}$$

Taking the logarithm for both sides of the likelihood function, we obtain the log-likelihood as follows:

$$\ln L = \sum_{i=0}^{r-1}\ln(\sum_{z_i=0}^{x_i} p(z_i)q(s, y_i)).$$

Using the logarithmic differentiation[1], we can calculate

$$\frac{d\binom{s}{y_i}}{ds} = \binom{s}{y_i}\sum_{j=0}^{y_i-1}\frac{1}{s-j}.$$

Hence,

$$\begin{aligned}\frac{dq(s, y_i)}{ds} &= \binom{s}{y_i}(\frac{1}{r})^{y_i}(1-\frac{1}{r})^{s-y_i}(\sum_{j=0}^{y_i-1}\frac{1}{s-j} + \ln(1-\frac{1}{r})) \\ &= q(s, y_i)(\sum_{j=0}^{y_i-1}\frac{1}{s-j} + \ln(1-\frac{1}{r})).\end{aligned}$$

Finally, we obtain

$$\frac{d\ln L}{ds} = \sum_{i=0}^{r-1}\frac{\sum_{z_i=0}^{x_i} p(z_i)q(s, y_i)(\sum_{j=0}^{y_i-1}\frac{1}{s-j} + \ln(1-\frac{1}{r}))}{\sum_{z_i=0}^{x_i} p(z_i)q(s, y_i)}. \tag{22}$$

We find that $\frac{d\ln L}{ds}$ is monotonically decreasing with respect to s. We can use binary search to efficiently find a value of s that makes $\frac{d\ln L}{ds} = 0$, which maximizes $\ln L$ and thereby L. Therefore, we obtain an estimator for s as follows:

$$\hat{s} = \arg\max_s\{\ln L\} = \{s|\frac{d\ln L}{ds} = 0\}. \tag{23}$$

[1]Since $[\ln(f)]' = \frac{f'}{f}$, we know $f' = f[\ln(f)]'$.

B. Analysis of $\hat{s}$

Following the analysis in Section V-C, $X_i \overset{\text{approx}}{\sim} N(\mu, \sigma^2)$. Hence, the probability density function of X_i is $f_{X_i}(x_i) = \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}$. Therefore, the likelihood function for observing $X_0 = x_0$, $X_1 = x_1$, ..., $X_{r-1} = x_{r-1}$ can also be written as

$$L(s; x_0, x_1, \ldots, x_{r-1}) = \prod_{i=0}^{r}\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}.$$

Taking the logarithm of the likelihood function, we have

$$\begin{aligned}\ln L &= \sum_{i=0}^{r-1}\ln(\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}), \\ &= \sum_{i=0}^{r-1} - \ln\sqrt{2\pi} - \ln\sigma - \frac{(x_i-\mu)^2}{2\sigma^2}.\end{aligned}$$

The first and second order derivatives of $\ln L$ with respect to s are

$$\frac{d\ln L}{ds} = \sum_{i=0}^{r-1} -\frac{r-1}{2r^2\sigma^2} + \frac{x_i-\mu}{r\sigma^2} + \frac{(x_i-\mu)^2(r-1)}{2r^2(\sigma^2)^2},$$

$$\begin{aligned}\frac{d^2\ln L}{ds^2} = \sum_{i=0}^{r-1}(&\frac{(r-1)^2}{2r^4(\sigma^2)^2} - \frac{1}{r^2\sigma^2} - \frac{2(r-1)(x_i-\mu)}{r^3(\sigma^2)^2} \\ &- \frac{(r-1)^2(x_i-\mu)^2}{r^4(\sigma^2)^3}),\end{aligned}$$

where we have used $\frac{d\mu}{ds} = \frac{1}{r}$, and $\frac{d\sigma^2}{ds} = \frac{r-1}{r^2}$. Hence,

$$\begin{aligned}E(\frac{d^2\ln L}{ds^2}) &= r(\frac{(r-1)^2}{2r^4(\sigma^2)^2} - \frac{1}{r^2\sigma^2} - \frac{(r-1)^2\sigma^2}{r^4(\sigma^2)^3}) \\ &= -\frac{2r\sigma^2 + (r-1)^2}{2r^3\sigma^4},\end{aligned}$$

since $E(x_i - \mu) = 0$ and $E(x_i - \mu)^2 = \sigma^2$. Hence, the fisher information [26] $I(s|x_0, x_2, \ldots, x_{r-1}) = -E(\frac{d^2\ln L}{ds^2}) = \frac{2r\sigma^2+(r-1)^2}{2r^3\sigma^4}$. According to the asymptotic properties of maximum likelihood estimators [26], we have

$$\hat{s} \xrightarrow{d} N(s, \frac{1}{I(s|x_0, x_2, \ldots, x_{r-1})}) = N(s, \frac{2r^3\sigma^4}{2r\sigma^2 + (r-1)^2}). \tag{24}$$

Therefore, the standard relative error is

$$StdErr(\frac{\hat{s}}{s}) = \frac{\sqrt{\frac{2r^3\sigma^4}{2r\sigma^2+(r-1)^2}}}{s}, \tag{25}$$

and the $1 - \alpha$ confidence interval for s is

$$\hat{s} \pm Z_{\frac{\alpha}{2}}\sqrt{\frac{2r^3\sigma^4}{2r\sigma^2 + (r-1)^2}}. \tag{26}$$

VII. Experiments

A. Experiment Setup

We have implemented the Counter Tree (CT) architecture with both CTE and CTM estimation methods, and the most related counter-based architectures randomized counter sharing with MLM (MLM provides more accurate estimates than CSE) [7], [9] and Counter Braids (CB) [5], [6]. We use experiments to compare the performance of CTE, CTM, MLM and CB. Prior work [7] has shown that the counter-based architectures remarkably outperforms the bitmap-based MSCBF. Hence, we only compare our counter architecture with other counter-based solutions, and exclude MSCBF due to page limitation of this paper. Without losing generality, we use TCP flows for presentation, and we have obtained similar results when carrying out experiments with other types of flows. The network trace we use was captured by Cisco's NetFlow at the main gateway of our university, which contains about 68 million TCP flows and 750 million packets. During each measurement period, we assume approximately 10 million packets are processed which are generated by about 1 million flows. The trace segment used for presentation contains 10,051,379 packets and 1,070,632 flows. Hence the average flow size is 9.39 packets/flow. We observe very similar results when different trace segments are processed.

We conduct two sets of experiments. The first set is used for comparing CT, MLM and CB. We vary the available memory space M from 0.25MB, 0.5MB, 1MB to 2MB, which translates to approximately 2bits/flow, 4bits/flow, 8bits/flow and 16bits/flow, respectively. For Counter Braids, we follow the same architecture adopted by [5]: A two-layer CB with status bits, and 3 hash functions at both layers. The layer-1 counters are 8 bits deep and the layer-2 counters are 56 bits deep. For MLM, we set the counter length to 6 bits, and the size of each storage vector to 100 [7]. For CT, we implement a tree with degree d fixed to 3. For fair comparison with MLM, we let the number of virtual counters in CT equal to the number of counters in MLM. Recall from (4) that $M' = \frac{2}{3}M$. Therefore, $b = \frac{2}{3} \times 6 = 4$ bits. In addition, we set the size r of each virtual counter array to 100. The results are presented in Section VII-B and Section VII-C. The second set of experiments aim at investigating the parameters' impacts on the performance of CT. We fix $M = 0.5$MB, and vary b, d and r with different values. The results are given in Section VII-D.

B. Processing Time for Encoding a Packet

The processing time for encoding a packet mainly results from memory accesses to read and write counters and the computations of hash values. A typical implementation of Counter Braids requires 3 hash functions on each layer, mapping each flow to the corresponding counters. To encode a packet, the router needs to read the 3 associated counters on the first layer, increment them by 1, and then write them back to the memory. If any of the 3 counters overflows, the router has to read and write another 3 counters on the second layer, which requires another 3 hash computations. Hence, the lower bounds of the number of memory accesses and the number of hash computations by CB are 6 and 3, respectively. In contrast, MLM aligns all counters on the same layer, and each packet is hashed to only one counter, which requires 2 memory access and 1 hash computation. CT also only requires 1 hash computation to determine the virtual counter for a packet. Recall that (9) gives an upper bound of amortized number of memory accesses by CT. When $b = 4$, the amortized number of memory accesses is bounded by $2 + \frac{1}{2^4-1} \approx 2.13$. In the first set of experiments, we record the average number of memory accesses and average number of hash computations for encoding a packet by CB, MLM and CT. The results are shown in Table I. We can see that CT is almost as efficient as MLM, and they achieve approximately 3× efficiency of CB. Moreover, the average number of memory accesses of CT decreases when more memory (counters) are available since each counter is shared by fewer flows, which reduces the overflows.

memory size (MB)	number of memory accesses			number of hash computations		
	CB	MLM	CT	CB	MLM	CT
0.25	6.01	2	2.09	3.01	1	1
0.5	6.01	2	2.06	3.00	1	1
1	6.01	2	2.03	3.00	1	1
2	6.01	2	2.02	3.00	1	1

TABLE I: Comparison of average processing time for encoding a packet by CB, MLM and CT.

C. Estimation Accuracy

We now present the estimation results by CB, MLM, CTE and CTM when different sizes of memory are available.

The estimation results of CB[2] are shown in Fig. 6 which includes four plots for different values of M. Each point in the plots represents an $(s, \hat{s})$ pair for a particular flow, where the x coordinate is the true flow size s and the y coordinate is the estimated flow size $\hat{s}$. The equality line, $y = x$, is presented for reference: The closer a point is to the equality line, the more accurate the estimate is. We can see that when a very tight memory $M = 0.25$MB is available, CB cannot produce any meaningful results. When $M = 0.5$MB and 1MB, CB generates positively biased results that are all above the equality line. When the available memory space increases to 16MB, CB can yield very accurate estimates as shown in the fourth plot.

Fig. 7 presents the estimation results of MLM. MLM can yield accurate estimates for small or moderate flows even under a tight memory space, e.g., $M = 0.25$MB. However, the counters large flows are mapped to may overflow and therefore cannot encode all packets of those large flows. As a result, MLM produces very negatively biased estimates for large flows. Although the increase of M can enlarge the estimation range of MLM to some extent, it does not address

[2]We find the estimation results provided by CB do not converge when the available memory is tight, so we terminate the process after 1000 iterations.

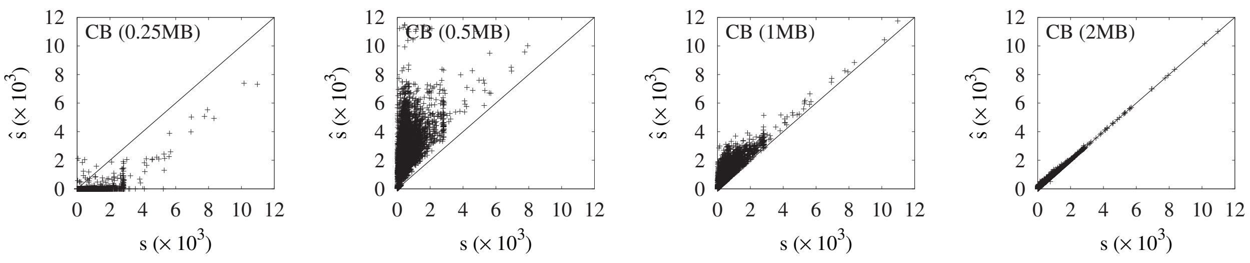

Fig. 6: • *First Plot*: estimation results by Counter Braids when $M = 0.25$MB. Each point in the plot represents an $(s, \hat{s})$ pair for a particular flow, where the x coordinate is the true flow size s and the y coordinate is the estimated flow size $\hat{s}$. The equality line, $y = x$, is presented for reference: A point closer to the equality line is more accurate. • *Second Plot*: estimation results by Counter Braids when $M = 0.5$MB. • *Third Plot*: estimation results by Counter Braids when $M = 1$MB. • *Fourth Plot*: estimation results by Counter Braids when $M = 2$MB.

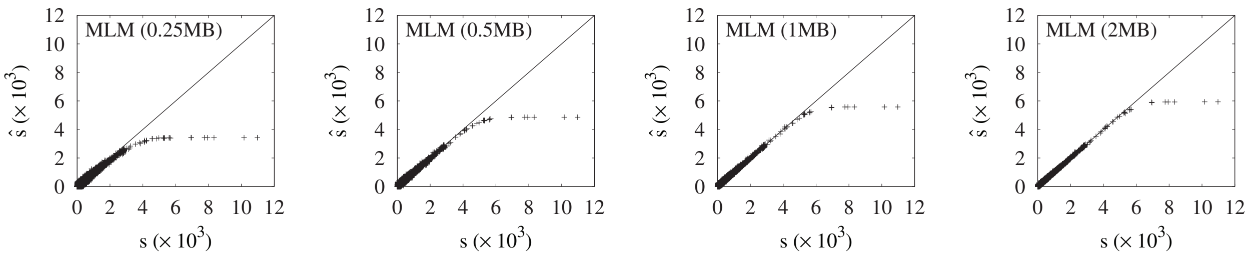

Fig. 7: Estimation results by MLM when $M = 0.25$MB, 0.5MB, 1MB, and 2MB, respectively.

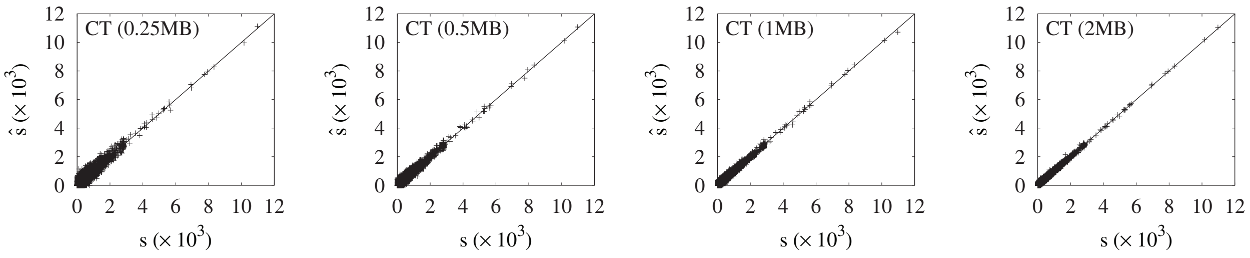

Fig. 8: Estimation results by CTE when $M = 0.25$MB, 0.5MB, 1MB, and 2MB, respectively.

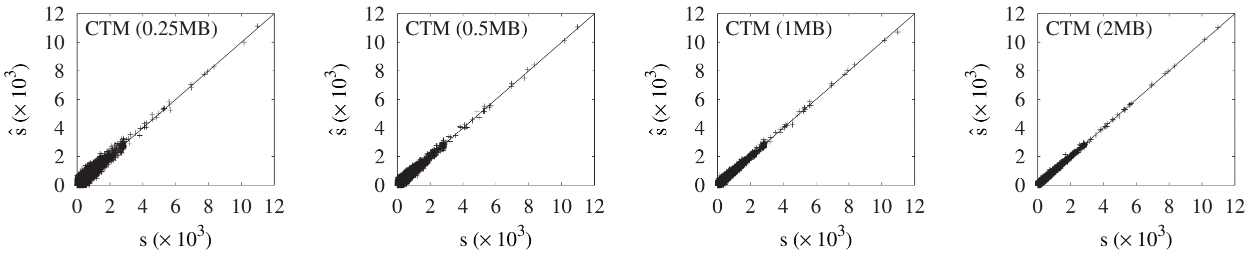

Fig. 9: Estimation results by CTM when $M = 0.25$MB, 0.5MB, 1MB, and 2MB, respectively.

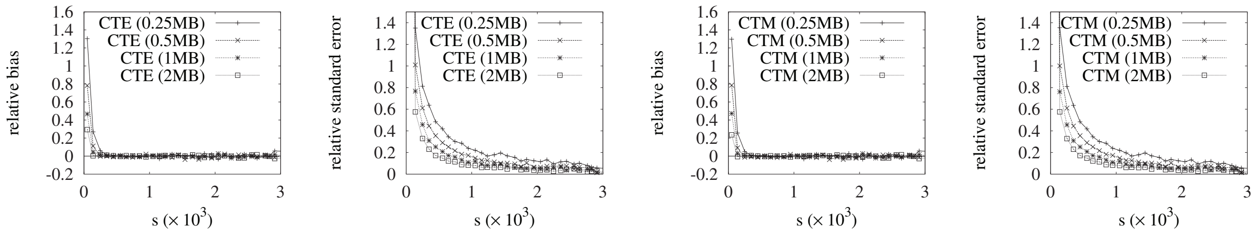

Fig. 10: • *First Plot*: the relative estimation bias $Bias(\frac{\hat{s}}{s})$ of CTE. • *Second Plot*: the relative standard error $StdErr(\frac{\hat{s}}{s})$ of CTE. • *Third Plot*: the relative estimation bias $Bias(\frac{\hat{s}}{s})$ of CTM. • *Fourth Plot*: the relative standard error $StdErr(\frac{\hat{s}}{s})$ of CTM.

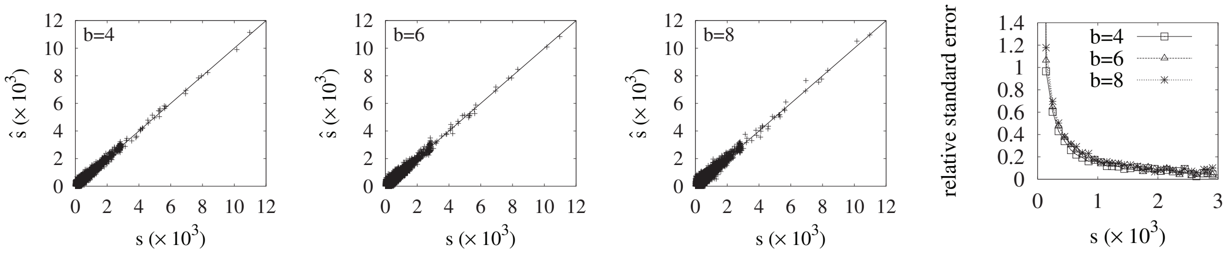

Fig. 11: Impact of b on the performance of CTE, where $M = 0.5$MB, $d = 3$, and $r = 100$. • *First Plot*: estimation results of CTE when $b = 4$. • *Second Plot*: estimation results of CTE when $b = 6$. • *Third Plot*: estimation results of CTE when $b = 8$. • *Fourth Plot*: the comparison of relative standard error when b = 4, 6, 8.

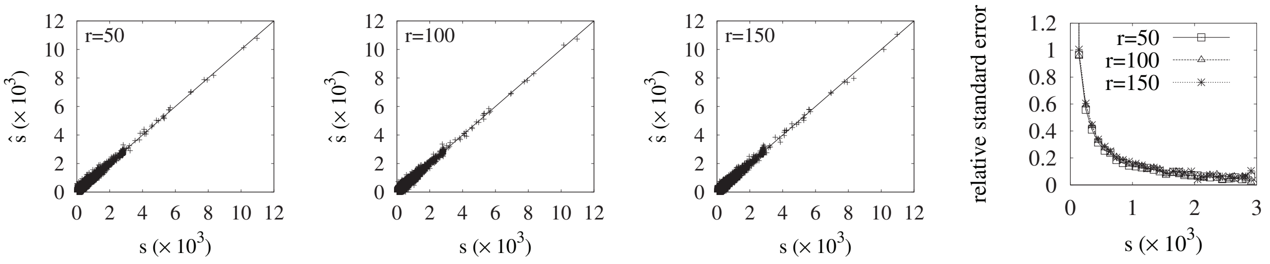

Fig. 12: Impact of r on the performance of CTE, where $M = 0.5$MB, $b = 4$ and $d = 3$.

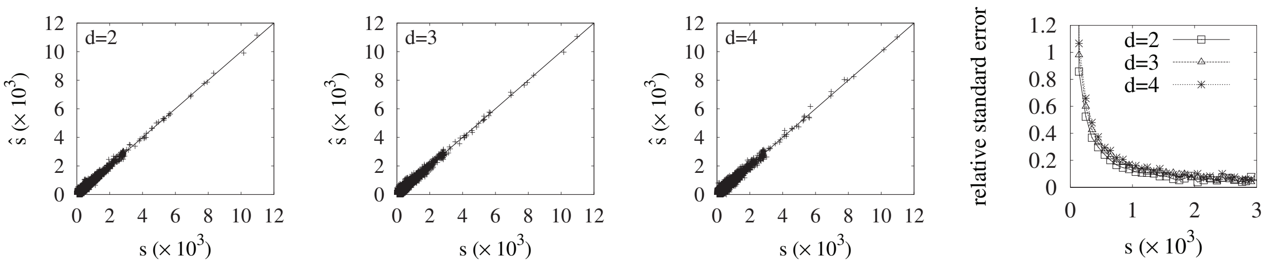

Fig. 13: Impact of d on the performance of CTE, where $M = 0.5$MB, $b = 4$ and $r = 100$.

the problem fundamentally. For example, the estimation range is about 6000 when $M = 16$MB.

The estimation results of CTE and CTM are given in Fig. 8 and Fig. 9, respectively. As expected, the employment of Counter Tree architecture significantly extends the counting range than MTM. Moreover, we observe that only the first two layers of the tree are used (the effective height of the tree is 2), implying that CT is capable of estimating much larger flows. Both CTE and CTM can yield very accurate estimates for all flows, including flows with very large sizes, even under a tight memory, e.g., 2 bits per flow. The estimates become more accurate when more memory space is available. The relative estimation bias $Bias(\frac{\hat{s}}{s})$ and the relative standard error $\frac{\sqrt{Var(\hat{s})}}{s}$ of CTE and CTM are presented in Fig. 10. We find that CTE and CTM in fact have comparable estimation accuracy.

In summary, CB does not suite for traffic measurement under a very tight memory, where it only produces unmeaningful results; MLM cannot be applied to estimate the sizes of large flows under a tight memory, which is a serious problem since it is hard to predict the sizes of the flows to be measured; In contrast, CT performs much better than CB under a tight memory, and it significantly extends the estimation range when compared with MLM.

D. Impact of b, r and d

We now vary the system parameters b, r and d to study their impacts on the performance of CT, while M is fixed to 0.5MB. The parameters are set as follows:

1) Impact of b: we fix $d = 3$, $r = 100$, and vary b from 4, 6 to 8.
2) Impact of r: we fix $b = 4$, $d = 3$, and vary r from 50, 100 to 150.
3) Impact of d: we fix $b = 4$, $r = 100$ and vary d from 2, 3 to 4.

We find that those parameters affect CTE and CTM in a similar way. Hence, we only present the estimation results of CTE in Fig. 11, Fig. 12, and Fig. 13. It is clear that the performance of CTE is not very sensitive to the change of b, r or d. Although the estimation accuracy of CTE deteriorates a little with the increase of b, r, or d as shown in the fourth plot of each figure, CTE still yields very accurate results. Since the increase of b, r or d makes each counter be shared by more flows, which in turn introduces more noise, the estimation accuracy degrades.

VIII. Related Work

We have elaborated on the directly related work of counter architectures in the introduction. In this section, we discuss the prior art for a related but different problem, called flow cardinality estimation, which is to estimate the number of *distinct* elements in each flow during a measurement period. A significant difference is that flow cardinality estimation needs to remove duplicate elements.

Bitmap [14] is a compact data structure that can be used for cardinality estimation. All bits are initialized to zeros. When an element arrives, it is hashed to a bit that will be set to one. Duplicate elements mapped to the same bit are automatically filtered out. At the end of a measurement period, the cardinality is estimated based on the size of the bitmap and the ratio of zeroes in the bitmap.

MultiResolutionBitmap [15] employs an array of bitmaps, each having a different sampling probability, to extend estimation range. Probabilistic Counting with Stochastic Averaging (PCSA) [16] (also known as FM sketch) maps each element to the ith (zero-based indexing) bit with a probability $\frac{1}{2^{i+1}}$. An FM sketch, also referred to as a *register* in the literature, can give an estimation up to 2^w, where w is the number of bits in the register.

LogLog [17] and HyperLogLog [18], [27] reduce the size of each register from 32 bits to 5 bits while retaining the same estimation range of 2^{32}. As a result, there will be many more registers available under the same memory constraint. Therefore, LogLog and HyperLogLog (HLL) significantly improve the estimation accuracy of PCSA.

IX. Conclusion

In this paper, we design a scalable counter architecture Counter Tree. We propose a two-dimensional sharing scheme, where each counter can be shared by multiple virtual counters and each virtual counter can be shared by multiple flows. As a result, Counter Tree significantly reduces memory requirement and extends estimation range. To encode a packet, Counter Tree only requires a little more than 2 memory accesses, which is asymptotically optimal. We propose two offline decoding methods to estimate flow sizes. The extensive experiments with real network trace demonstrate that our methods can yield very accurate estimates even under an extremely tight memory space, e.g., 2 bits per flow. In our future work, we will investigate the performance of Counter Tree when the flows sizes follow different distributions. In addition, we will explore the theoretic upper bound of estimation accuracy of counter architectures under a given memory space.

X. Acknowledgments

This work was supported in part by the National Science Foundation under grant NeTS-1115548.

References

[1] D. Shah, S. Iyer, B. Prabhakar, and N. McKeown, "Analysis of a statistics counter architecture," in *Hot Interconnects 9, 2001*. IEEE, 2001, pp. 107–111.
[2] S. Ramabhadran and G. Varghese, "Efficient implementation of a statistics counter architecture," in *ACM SIGMETRICS Performance Evaluation Review*, 2003, vol. 31, pp. 261–271.
[3] Q. Zhao, J. Xu, and Z. Liu, "Design of a novel statistics counter architecture with optimal space and time efficiency," *ACM SIGMETRICS Performance Evaluation Review*, vol. 34, no. 1, pp. 323–334, 2006.
[4] A. Kumar, J. Xu, and J. Wang, "Space-code bloom filter for efficient per-flow traffic measurement," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 12, pp. 2327–2339, 2006.
[5] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter Braids: A Novel Counter Architecture for Per-Flow Measurement," *Proc. of ACM SIGMETRICS*, June 2008.
[6] Y. Lu and B. Prabhakar, "Robust Counting Via Counter Braids: An Error-Resilient Network Measurement Architecture," *Proc. of IEEE INFOCOM*, April 2009.

[7] T. Li, S. Chen, and Y. Ling, "Fast and Compact Per-Flow Traffic Measurement through Randomized Counter Sharing," *Proc. of IEEE INFOCOM*, pp. 1799–1807, April 2011.

[8] T. Li, S. Chen, and Y. Qiao, "Origin-Destination Flow Measurement in High-Speed Networks," *Proc. of IEEE INFOCOM*, pp. 2526–2530, 2012.

[9] T. Li, S. Chen, and Y. Ling, "Per-flow traffic measurement through randomized counter sharing," *IEEE/ACM Transactions on Networking*, vol. 20, no. 5, pp. 1622–1634, 2012.

[10] N. Wang, K. H. Ho, G. Pavlou, and M. Howarth, "An Overview of Routing Optimization for Internet Traffic Engineering," *IEEE Communications Surveys and Tutorials*, vol. 10, pp. 36 – 56, 2008.

[11] S. Chen and K. Nahrstedt, "Maxmin Fair Routing in Connection-Oriented Networks," *Proc. of Euro-PDS*, pp. 163–168, 1998.

[12] S. Chen, Y. Deng, P. C. Attie, and W. Sun, "Optimal Deadlock Detection in Distributed Systems Based on Locally Constructed Wait-for Graphs," *Proc. of IEEE ICDCS*, pp. 613–619, 1996.

[13] K. C. Claffy, H. W. Braun, and G. C. Polyzos, "A Parameterizable Methodology for Internet Traffic Flow Profiling," *IEEE Journal on Selected Areas in Communications*, vol. 13, pp. 1481–1494, 1995.

[14] K. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A Linear-time Probabilistic Counting Algorithm for Database Applications," *ACM Transactions on Database Systems*, vol. 15, no. 2, pp. 208–229, June 1990.

[15] C. Estan, G. Varghese, and M. Fish, "Bitmap algorithms for counting active flows on high-speed links," *IEEE/ACM Transactions on Networking*, vol. 14, no. 5, pp. 925–937, 2006.

[16] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for database applications," *Journal of Computer and System Sciences*, vol. 31, pp. 182–209, September 1985.

[17] M. Durand and P. Flajolet, "Loglog counting of large cardinalities," *Proc. of European Symposia on Algorithms*, pp. 605–617, 2003.

[18] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm," *Proc. of AOFA*, pp. 127–146, 2007.

[19] *NEC and Corning achieve petabit optical transmission*, Available at http://optics.org/news/4/1/29.

[20] N. Duffield, C. Lund, and M. Thorup, "Estimating Flow Distributions from Sampled Flow Statistics," *Proc. of ACM SIGCOMM*, October 2003.

[21] A. Kumar, M. Sung, J. Xu, and J. Wang, "Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Size Distribution," *Proc. of ACM SIGMETRICS*, June 2004.

[22] N. Kamiyama and T. Mori, "Simple and Accurate Identification of High-rate Flows by Packet Sampling," *Proc. of IEEE INFOCOM*, April 2006.

[23] C. Estan and G. Varghese, "New Directions in Traffic Measurement and Accounting," *Proc. of ACM SIGCOMM*, August 2002.

[24] Y Qiao, T. Li, and S. Chen, "One Memory Access Bloom Filters and Their Generalization," *Proc. of IEEE INFOCOM*, pp. 1745–1753, 2011.

[25] W. Fang and L. Peterson, "Inter-as traffic patterns and their implications," in *Proc. of Global Telecommunications Conference*. IEEE, 1999, vol. 3, pp. 1859–1868.

[26] E. Lehmann and G. Casella, "Theory of Point Estimation," *Springer Press*, 1998.

[27] Q. Xiao, S. Chen, M. Chen, and Y. Ying, "Hyper-Compact Virtual Estimators for Big Network Data Based on Register Sharing," *Proc. of ACM SIGMETRICS*, , no. 417-428, 2015.