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

Min Chen, Shigang Chen, *Fellow, IEEE*, and Zhiping Cai, *Member, IEEE*

***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 traffic engineering, classification of routing distribution or network usage pattern, service provision, anomaly detection, and network forensics. In order to keep up with the high throughput of modern routers or switches, the online module for per-flow traffic measurement should use high-bandwidth SRAM that allows fast memory accesses. Due to 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 approach to estimate the flow sizes using counter architectures that can fit into tight SRAM. However, existing counter architectures have limitations, either still requiring considerable SRAM space or having a small estimation range. In this paper, we design a scalable counter architecture called Counter Tree, which leverages a 2-D counter sharing scheme to achieve far better memory efficiency and in the meantime extend estimation range significantly. Furthermore, we improve the performance of Counter Tree by adding a status bit to each counter. Extensive experiments with real network traces demonstrate that our counter architecture can produce accurate estimates for flows of all sizes under very tight memory space.**

***Index Terms*— Traffic measurement, flow size, counter architecture, virtual counter, counter sharing.**

## I. Introduction

PER-FLOW traffic measurement is one of the fundamental problems in network traffic measurement [1]–[11]. 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, http flows, or any user-defined logical flows. Each flow is uniquely identified by its flow label — for example, the flow labels for per-source flows are the source addresses. Per-flow traffic measurement has many important applications in traffic engineering, classification of routing distribution or network usage pattern, service provision, anomaly detection and network forensics. For example, ISPs can use the per-flow information to optimize traffic routing in their backbone networks to reduce congestion; SDN networks may decide which flows to be re-routed on which paths based on the flow sizes and the bandwidth availability on those paths; per-flow measurement can also help determine the distribution of traffic transmitted in the networks and the characteristics of traffic sent from a particular source or destined to a particular address; network administrators can perform per-source traffic measurement to identify scanners or estimate the scanning rates of malicious hosts.

We stress that per-flow traffic measurement significantly differs from a related problem called *flow cardinality estimation* [12]–[16], 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 under measurement. We will discuss flow cardinality estimation with more details in the related work section.

*Technical Challenge:* 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 (the time needed to locate and retrieve/update a single piece of information) is approximately 50 to 150 nanoseconds [17], cannot keep up with the dramatically increasing line speed (NEC and Corning achieved a transmission rate of 1.05 Petabit/sec using a 12-core fiber design [18]). On the contrary, SRAM with much smaller access time has low density, so large SRAM is expensive and difficult to implement on chip. Moreover, the limited on-chip SRAM is shared by different functions, such as routing, scheduling, traffic measurement, and security. Even for traffic measurement alone, there can be multiple functions performed concurrently for different purposes, each requiring some SRAM. Therefore, the SRAM dedicated for per-flow traffic measurement can be very small, necessitating memory-efficient measurement approaches. In addition, we need to minimize the processing overhead required by 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 switches.

With high-speed networks routinely carrying large numbers of flows, it is often 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

Manuscript received July 16, 2015; revised February 27, 2016 and September 27, 2016; accepted October 9, 2016; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor A. Bremler-Barr. Date of publication November 18, 2016; date of current version April 14, 2017. This work was supported in part by the National Science Foundation under Grant NeTS 1115548 and in part by the National Natural Science Foundation of China under Grant 61379145, Grant 61232016, and Grant U1405254.

M. Chen and S. Chen are with the Department of Computer and Information Science Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: min@cise.ufl.edu; sgchen@cise.ufl.edu).

Z. Cai is with the College of Computer, National University of Defense Technology, Changsha 410073, China (e-mail: zpcai@nudt.edu.cn).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org. This consists of a 45.13-KB PDF containing Appendixes A and B.

Digital Object Identifier 10.1109/TNET.2016.2621159

SRAM are incremented at high speed, and occasionally written back to larger counters in DRAM. However, the hybrid architecture incurs costly SRAM-to-DRAM updates. Furthermore, the flow-to-counter association requires considerable extra SRAM [5].

*Prior Art and Limitation:* To fit the measurement module in tight SRAM, some schemes only provide the distribution of flow sizes [19], [20], or measure the sizes of large flows [21], [22]. Other work lessens the space requirement by estimating the sizes of the flows instead of counting their exact sizes. The state-of-the-art estimation approaches include bitmap-based MSCBF, Counter Braids, and randomized counter sharing, which will be briefly reviewed below.

The *Multiresolution Space-Code Bloom Filter* (MSCBF) [4] employs multiple Bloom filters [23] 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]. TinyTable [24] is a novel hash table based data structure that represents multiset membership. It improves the query and update efficiency of Bloom filters. However, for per-flow traffic measurement, it still requires a high memory cost, which is tens of bits per flow [24].

The *Counter Braids* (CB) [5], [6] are a counting 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 record a packet, it hashes the flow label of the packet to three counters, 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, CB 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. Second, it performs 6 (occasionally 12) memory accesses to record one packet. Third, CB does not support efficient on-demand queries of selected flows. The sizes of all flows must be computed together.

The scheme of *randomized counter sharing* [7] was proposed to further reduce the memory requirement and processing overhead 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 recording a packet of a certain flow, it maps the flow label to a counter in the flow's storage vector, and increments the counter by one. This scheme requires only 2 memory accesses for recording 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 of packets 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 full version [8], some approaches were provided to extend the estimation range, which however cannot address the issue fundamentally. The first approach is to increase the number of bits in each counter. This approach degrades estimation accuracy because accuracy is related to the total number of counters available in the system [8] and increasing the counter size reduces the number of counters under the same memory constraint. The second approach employs a sampling module. Each arriving packet is sampled with a probability $p$ before being recorded 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 packets are hardly 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 on-chip SRAM-based counter architecture, with new contributions summarized as follows:

1) The main contribution of Counter Tree is a two-dimensional counter sharing scheme, where physical counters are shared to compose different virtual counters, which are in turn shared among different flows. This new scheme is able to work with very tight on-chip space when Counter Braids [5] no longer work, and in the meantime solve the estimation range limitation of the randomized counter sharing scheme [7].
2) Counter Tree has low processing overhead. Recording a packet only requires a little more than 2 memory accesses on average, which is near optimal.
3) It supports efficient query on the size of an arbitrary flow, without having to compute the sizes of other flows as Counter Braids do. Two offline methods are proposed for flow-size estimation. Extensive experiments with real network traces demonstrate that both methods can generate good estimation results even with very tight memory.

The rest of the paper is organized as follows. Section II defines performance metrics used in the paper. Section III presents the design of Counter Tree architecture. Section IV shows how to record a packet in the Counter Tree. Section V and Section VI propose and analyze two estimation methods. Section VII describes the enhanced Counter Tree architecture. Section VIII evaluates the proposed Counter Tree architectures through experiments. The related work is given in Section IX. Sections X draws the conclusion.

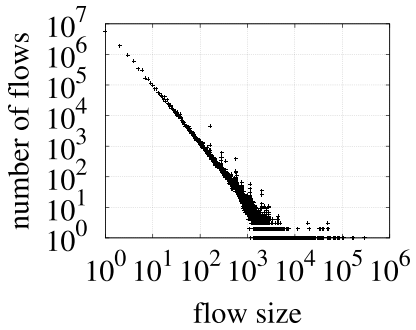



Fig. 1. Flow size distribution, where each point represents the number ($y$ coordinate) of flows that have a particular size ($x$ coordinate).

## 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 on-chip space, we want to use as small memory as possible to achieve per-flow traffic measurement. In the sequel we refer to SRAM simply as memory. This focuses on the memory requirement for implementing the counter architectures. 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 Overhead:* To keep up with the line speed, the processing overhead for recording a packet should be small, such that the implementation of the measurement module will not cause a performance bottleneck. In most counter architectures [4], [5], [7], the processing overhead for recording a packet mainly results from memory accesses and hash computations.

*Estimation Accuracy:* With a given memory space, we want the estimates of flow sizes to be as accurate as possible. Let the true size of a flow be $s$ and the estimated size be $\hat{s}$. We use the relative bias $Bias(\frac{\hat{s}}{s})$ and relative standard error $StdErr(\frac{\hat{s}}{s})$ to evaluate the estimation accuracy, 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

Many studies have revealed 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 the top 15% of the destination prefixes (per-destination flows) account for over 95% of the byte traffic. As an example, we use a network trace obtained from the main gateway of University of Florida, which contains about 68 million TCP flows and 750 million packets. The distribution of flow sizes is shown 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 or medium 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 have to be set large according to the maximum flow size to handle the worse case. However, most flows will turn out to be small or medium. For flows of a few packets, most of the counter bits will be wasted. This observation motivates us to save memory by utilizing the waste.

### B. Two-Dimensional Counter Sharing

To reduce the memory waste caused by small or medium 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. The available memory space is divided into many small physical counters, which are logically organized in a tree structure with multiple layers. Each flow is pseudo-randomly mapped to a number of counters at the bottom layer, and the flow will be recorded by these counters. As different flows randomly choose their counters, multiple flows may be mapped to the same counter, which results in horizontal counter sharing. Whenever a bottom-layer counter is overflowed, its parent counter in the tree is increased by one. When that parent is overflowed, its next-layer parent is increased by one. There are fewer parents than children at each layer in the tree. Hence, children counters share parent counters as their high-order bits, which is vertical counter sharing. Note that horizontal counter sharing happens at each layer (not just the bottom layer). The reason is that as the bottom-layer counters are overflowed to higher layers, each higher-layer counter may again be shared by multiple flows to record their packets.

In *horizontal counter sharing*, each counter is shared by multiple flows. The rationale is to let large flows *borrow* memory from small or medium flows that will not fully use their counters. But each counter will have to store the combined size of multiple flows, particularly if the number of counters is much smaller than the number of flows. To alleviate this problem, the CountMin approach [26] maps each flow to multiple counters and use the smallest counter value as the flow size. The problems are that each packet of a flow will result in multiple counter updates (which reduces the processing throughput multiple folds) and that the smallest counter may still be the sum of multiple flow sizes, causing positive bias in estimation. Our new design in the next subsection will keep the benefit of counter sharing, while ensuring that approximately one counter is updated per packet and in the meantime avoiding the positive bias in estimation. The idea of our design is to split each flow to multiple counters through random mapping, and each counter can therefore be shared by different flows.

Horizontal counter sharing allows some bits wasted by small or medium flows to be used by large flows. However, since small flows account for a dominant percentage of network flows, many bits in counters occupied only by small flows can still be wasted. This observation leads to the idea of *vertical counter sharing* below, which allows the more significant bits (i.e., higher-order bits) to be shared by more flows.

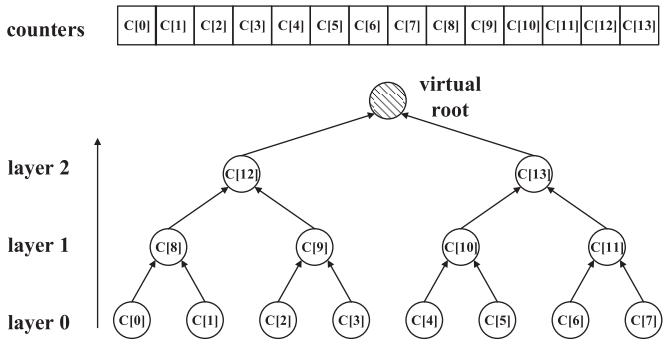



Fig. 2. An example of organizing counters into a binary tree.

We introduce the concept of *virtual counters*, each of which is the concatenation of multiple small physical counters, which are also called the *component counters* of the virtual counter. Physical counters do not share their bits, but virtual counters share bits by sharing their component counters, particularly those representing more significant bits in virtual counters. In essence, vertical counter sharing implements seamless dynamic memory allocation based on flow sizes, without having to allocate or deallocate physical counters of different sizes on the fly (which is too costly to implement on chip). Suppose we have three physical 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. These three component counters only require 9 bits to successfully record $f$ and $g$.

The scheme of two-dimensional counter sharing contributes to significant memory saving, but it also introduces noise among virtual counters due to space sharing. Fortunately, we can employ statistical tools to remove such noise as we will show shortly. In the sequel, when we use the word "counter" without preceding it with "virtual", we mean a physical (component) counter by default unless the context clearly suggests otherwise.

### C. Counter Tree Architecture

We design a Counter Tree architecture to implement two-dimensional counter sharing. The architecture can be used to record flows of all sizes (small, medium or large). Given a memory space of $M$ bits, we divide it into small counters, each consisting of $b$ bits. We organize those counters into a tree structure from the bottom up. Let the leaf nodes be the layer 0 which consists of $m$ counters. The degree of each non-leaf node is $d$. Therefore, the number of counters on a upper layer is $\frac{1}{d}$ of it lower layer. Denote the height of the tree by $h$, with layers indexed from 0 to $h-1$. We have the following constraint

$$\sum_{j=0}^{h-1} \frac{m}{d^j} \times b \leq M. \tag{3}$$

Therefore,

$$m \leq \frac{d^{h-1}M}{(d^h-1)b}. \tag{4}$$

If the $(h-1)$th layer contains more than one counter, namely, $\frac{m}{d^{h-1}} > 1$, we put an extra node as the root of the tree, called *virtual root*. Fig. 2 gives an example of organizing the 14 counters into a binary tree with 3 layers, where $m = 8$ and $d = 2$. Starting from a leaf node $C[i]$ $(0 \leq i < m)$, the $h$ counters along the path to the root form a virtual counter, denoted as $V[i]$. As a result, there will be $m$ virtual counters in total, denoted as an array $V$.

Starting from $C[i]$ at layer 0, the counter at layer $j$ $(0 \leq j < h)$ that $V[i]$ will include, denoted by $V[i][j]$, is

$$V[i][j] = C[\lfloor \frac{i}{d^j} \rfloor + \sum_{t=1}^{j} \frac{m}{d^{t-1}}]. \tag{5}$$

Therefore, the Counter Tree in Fig. 2 can yield 8 virtual counters as shown in the upper half of Fig. 3. We can see that a counter located at a higher layer (which corresponds to more significant bits in a virtual counter) is shared by more virtual counters but will be used only by large flows. This embodies the idea of vertical counter sharing. Later we will show that each virtual counter can be shared by multiple flows, corresponding to the idea of horizontal counter sharing.

We note that multiple layers of counters are also used by Counter Braids [5], [6] under a different design. For Counter Braids, each flow is randomly mapped to $u$ counters at the bottom layer, where $u$ may be 3. Any packet of a flow will cause all $u$ counters of the flow to increase by one. Hence, each counter records the total number of packets for all flows that are mapped to it. That is, each counter represents a linear equation on the sizes of some flows (that are mapped to the counter). When there are enough counters, there will be enough linear equations, which we can solve (by the method of message passing in [5]) for the sizes of all flows. But this approach requires updating $u$ counters per packet and it needs a sufficient number of counters. What about the memory space is too tight and thus the number of counters is insufficient? Our method can work under such tight space where Counter Braids no longer work. In Counter Tree, each flow is randomly mapped to $r$ counters at the bottom layer, where $r$ should be large, e.g., in hundreds. Any packet of a flow will cause one of the $r$ counters to increase by one. Therefore, each counter no longer represents a linear equation on the sizes of some flows, which also means that the estimation method of Counter Braids cannot be applied here. Instead, we view the $r$ counters of a flow $f$ as a whole, whose sum carries both the size of flow $f$ and the noise from other flows due to counter sharing. We do not know exactly who those other flows are, but nevertheless the noise can be statistically measured and removed.

### D. Counting Range

The counting range of each virtual counter is $2^{bh}$. To extend the counting range, one way is to increase $b$. However, larger $b$ means fewer counters are available, e.g., if we double $b$, the number of available counters will be reduced by half. Moreover, if $b$ is set overly large, some bits in each counter will be wasted. Alternatively, we can increase $h$ for the purpose of extending the counting range, which is more memory efficient as we will show shortly. Suppose $M'$ bits are 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

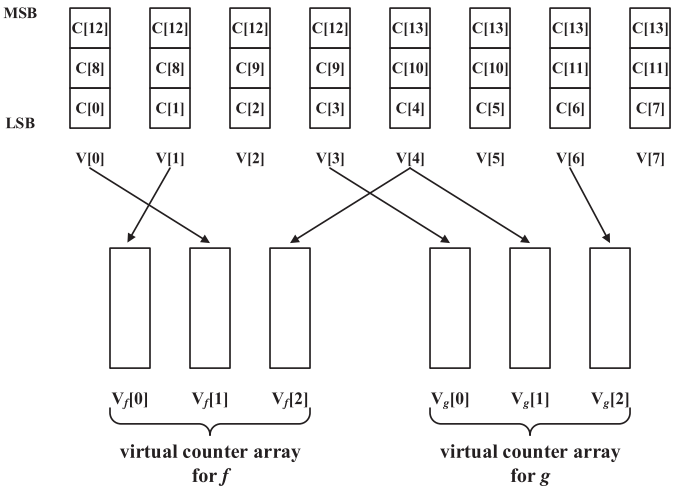



Fig. 3. Virtual counters and virtual counter arrays for flows. The virtual counter array for a particular flow consists of multiple counters pseudorandomly chosen from the counters.

the number of counters on the $j$th layers is reduced to $\frac{1}{d}$ of the $(j-1)$th layer, the following memory constraint should hold

$$\sum_{j=0}^{h-1} \frac{M'}{d^j} \leq M.$$

Since $\sum_{j=0}^{h-1} \frac{M'}{d^j} < \frac{d}{d-1}M'$, and we have

$$M' > \frac{d-1}{d}M, \tag{6}$$

which means that only $\frac{1}{d}$ of the memory needs to be reserved for non-leaf counters. Therefore, each virtual counter can have up to $b \times (\log_d \frac{M'}{b} + 1)$ bits, which translates to a counting range of $2^{b(\log_d \frac{M'}{b}+1)}$. In contrast, the counting range of each counter is only $2^b - 1$. This means that as long as we spare $\frac{1}{d}M$ memory for upper layers and set $M' = \frac{d-1}{d}M$, we can extend the counting range from $2^b - 1$ to $2^{b(\log_d \frac{M'}{b}+1)}$. The randomized counter sharing scheme [8] is a special case of Counter Tree with $h = 1$. Since it only records flows by one-layer physical counters, the estimation range of randomized counter sharing scheme is very limited, whereas the estimation range of Counter Tree with multiple layers is much larger. As an example, suppose $M$ = 1MB, $b = 4$, $d = 2$, and $M'$ is therefore 0.5MB. 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 significantly extend the estimation range to accommodate large flows.

## IV. Online Packet Recording

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

### A. Recording

Consider an arbitrary flow $f$. We pseudorandomly choose $r$ ($r \ll m$) 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. The $i$th 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 $0 \leq i < r$ 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. Since $r \ll m$, the probability that $r$ distinct virtual counters are selected by the hash functions to form the virtual counter array of $f$ is $\frac{m(m-1)(m-r+1)}{m^r} \approx 1$. 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. We point out that our design does not limit the number of flows to be supported. There can be many more flows than the number of virtual vectors, $m$.

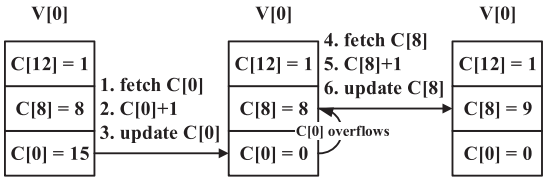



Fig. 4. The process for recording a packet in a virtual counter.

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 $f$, chooses a virtual counter from $V_f$ uniformly at random, 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 $V[u] \leftarrow V[u] + 1$. Note that the update of $V[u]$ may involve the updates of multiple counters. Based on (5), the router first fetches counter $C[u]$ from memory and increases it by 1. If $C[u]$ does not overflow, the recording for this packet is done. Otherwise, the router further fetches $C[\lfloor \frac{u}{d} \rfloor + m]$, and adds the overflowed 1 to $C[\lfloor \frac{u}{d} \rfloor + m]$. The process continues until no overflow happens or the counter on the highest layer has been reached. In the latter case, the virtual counter is overflowed beyond the upper bound that it is designed to handle. Fig. 4 gives an example of the online recording process for a packet of $f$. Suppose $b = 4$ (i.e., the counting range of each counter is 15), $h = 3$, and $V[0]$ is chosen for recording that packet. The router first fetches $C[0]$ whose 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 assigns $C[8] \leftarrow C[8] + 1$. Since $C[8] = 10$ does overflow, the router writes it back and the recording process terminates.

### B. Number of Memory Accesses

To record 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 recording a packet is 2. In the worst case, the router needs to update $h$

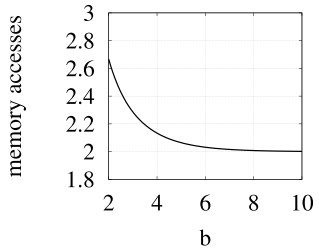



Fig. 5. Amortized number of memory accesses per packet with respect to $b$.

counters, which requires $2h$ memory accesses. The good thing is that the router needs to fetch another counter only when the current counter overflows, which happens after recording at least $2^b$ packets. Hence, the amortized number of memory accesses per packet is much smaller than the worst case. We have the following theorem:

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

*Proof:* Suppose $n$ packets are recorded. Each packet causes one counter at layer 0 to be updated, requiring 2 memory accesses. In total, there are $2n$ memory accesses at layer 0. The number of counter overflows at layer 0 is at most $\lfloor \frac{n}{2^b} \rfloor \leq \frac{n}{2^b}$, which means that counters at layer 1 will be updated for no more than $\frac{n}{2^b}$ times, requiring no more than $\frac{2n}{2^b}$ memory accesses. By simple induction, we know that the number of memory accesses at layer $j$ is no more than $\frac{2n}{2^{jb}}$. Hence, the total number of memory accesses over all layers is no more than

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

Hence, the amortized number of memory accesses per packet is no more than

$$\frac{2n(1+\frac{1}{2^b-1})}{n} = 2 + \frac{2}{2^b-1}. \tag{9}$$

The upper bound is tight when $n \equiv 0 \pmod{2^{(h-1)b}}$ and exactly $\frac{n}{2^{(j+1)b}}$ overflows happen at the $j$th layer, $0 \leq j < h-1$. ∎

Fig. 5 shows the upper bound of the amortized number of memory accesses for recording a packet with respect to $b$. We find that it quickly converges to 2 (the lower bound) with the increase of $b$. In contrast, Counter Braids need to perform at least $2u$ memory accesses since each packet is recorded by $u$ counters on each layer, where $u$ can be 3.

The non-deterministic counter access time per packet may introduce stalls into the datapath pipeline, resulting in reduced datapath bandwidth. Counter Braids [5], [6] and any other counter architectures with variable access time [3] face the same problem. The key issue is how frequent the variable access time will occur. In case that access time for most packets is constant but only varies for a tiny fraction of packets, the impact on datapath bandwidth will be limited, particularly if we are able to reduce the chance of variable access time to an arbitrarily small level through a system parameter. This is the case for Counter Tree. More specifically, the overflow of a layer-0 counter occurs once every $2^b$ packets. On average, one out of $2^b$ packets has longer access time because a counter at the higher layer is involved, but the other $2^b - 1$ packets have constant access time because they do not cause counter overflow. The fraction of all packets that have longer access time is $\frac{1}{2^b}$, which can be exponentially reduced by increasing the value of b, i.e., the number of bits in a counter.

## V. Counter Tree Based Estimation

After the measurement period, offline estimation should be performed to recover flow sizes from the Counter Tree. Counter Tree estimates flow sizes through some statistical methods, and it supports efficient query on the size of an arbitrary flow, without having to compute the sizes of other flows as Counter Braids do (the computation overhead is high). In this section, we first propose and analyze the Counter Tree based Estimation (CTE) method.

### A. CTE method

Consider the $i$th 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 $V[u]$ records some 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 packets in $f$ recorded by $V[u]$, we need to figure out how to remove such noise.

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 (5). Consider the subtree rooted at $C[v]$, denoted as $T$, consisting of $d^{h-1}$ leaf nodes, which correspond to $d^{h-1}$ virtual counters. Let $k = d^{h-1}$. Due to counter sharing, flows mapped to those $k$ virtual counters may introduce noise to $V[u]$. We treat $T$ as an aggregate, called a *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 $C[12] \times 2^{2b} + (C[8] + C[9]) \times 2^b + (C[0] + C[1] + C[2] + C[3])$. Note that the total number $n$ of packets in all flows can be obtained from the entire Counter Tree in a similar way. 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, $X_i = Y_i + Z_i$.

Let $s$ be the true flow size of $f$ during the measurement period. Assume there is a large number of flows, $n$ is large, the size of each flow is negligibly small when comparing with $n$, $r$ is much larger than 1, and $r \ll m$.

Flow $f$ has $r$ virtual counters (including $V[u]$) in its virtual counter array. Each packet of $f$ has a probability $\frac{1}{r}$ to be mapped to $V[u]$ and increment it by one. Therefore, $Y_i$ follows a binomial distribution:

$$Y_i \sim B(s, \frac{1}{r}). \tag{10}$$

Consider an arbitrary packet belonging to a different flow $g$. The probability for the virtual counter array of $g$ to include a particular virtual counter in $T$ is approximately

$1-\frac{\binom{m-1}{r}}{\binom{m}{r}}=\frac{r}{m}$. When that happens, the conditional probability for this particular virtual counter to record the packet is approximately $\frac{1}{r}$. Combining the above analysis, the probability for this particular virtual counter to record the packet is approximately $\frac{r}{m}\times\frac{1}{r}=\frac{1}{m}$. Since there are $k$ virtual counters in $T$, the probability that $T$ records the packet is $(1-\frac{1}{m})^k\approx\frac{k}{m}$ for $k\ll m$. There are $n-s$ packets outside of $f$. Since there are numerous flows under measurement, the total number $n$ of packets can be much larger than the size $s$ of any single flow, even for large flows, as we observe in our traffic traces. With $s\ll n$ and $n-s\approx n$, $Z_i$ roughly follows a binomial distribution

$$Z_i\sim B(n,\frac{k}{m}). \tag{11}$$

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,

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

Hence, we have

$$s=rE(X_i)-\frac{nkr}{m}. \tag{12}$$

We do not know the exact value of $E(X_i)$, but we have the $r$ instance values, also denoted as $X_i$, $0\le i<r$, that can be directly counted from the Counter Tree as subtree counters. Replacing $E(X_i)$ with the measured average $\frac{\sum_{i=0}^{r-1}X_i}{r}$, we obtain an estimate of $s$, denoted as $\hat{s}$, as follows:

$$\hat{s}=\sum_{i=0}^{r-1}X_i-\frac{nkr}{m}, \tag{13}$$

where the first term is the number of all packets recorded by the $r$ subtree counters and the second term captures the average noise presented in $r$ counters.

### B. Analysis of $\hat{s}$

Since $Y_i$ and $Z_i$ follow binomial distributions, we have

$$Var(Y_i)=\frac{s}{r}(1-\frac{1}{r}),\quad Var(Z_i)=\frac{nk}{m}(1-\frac{k}{m}). \tag{14}$$

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

$$\begin{aligned}Var(X_i)&=Var(Y_i)+Var(Z_i)+2Cov(Y_i,Z_i)\\&=\frac{s}{r}(1-\frac{1}{r})+\frac{nk}{m}(1-\frac{k}{m}).\end{aligned} \tag{15}$$

Therefore,

$$E(\hat{s})=E(\sum_{i=0}^{r-1}X_i)-\frac{ndr}{m}=r(\frac{nkr}{m})-\frac{nkr}{m}=s, \tag{16}$$

which means the estimator $\hat{s}$ is unbiased. In addition, we have

$$\begin{aligned}Var(\hat{s})&=Var(\sum_{i=0}^{r-1}X_i)=r^2Var(X_i)\\&=s(r-1)+\frac{nr^2b(d^h-1)}{M}(1-\frac{b(d^h-1)}{M}),\end{aligned} \tag{17}$$

where we have used $m=\frac{d^{h-1}M}{b(d^h-1)}$ and $k=d^{h-1}$. Hence, the standard error of the ratio $\frac{\hat{s}}{s}$ is

$$StdErr(\frac{\hat{s}}{s})=\frac{\sqrt{s(r-1)+\frac{nr^2b(d^h-1)}{M}(1-\frac{b(d^h-1)}{M})}}{s}. \tag{18}$$

When $n$ is sufficiently large, the binomial distribution, $Z_i\sim B(n,\frac{k}{m})$, can be approximated by a normal distribution, $N(\frac{nk}{m},\frac{nk}{m}(1-\frac{k}{m}))$. Similarly, $Y_i\overset{\text{approx}}{\sim}N(\frac{s}{r},\frac{s}{r}(1-\frac{1}{r}))$. Since the linear combination of independent normal distributions also follows 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 (13), we have

$$\hat{s}\overset{\text{approx}}{\sim}N(s,s(r-1)+\frac{nkr^2}{m}(1-\frac{k}{m})). \tag{19}$$

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

$$\hat{s}\pm Z_\alpha\sqrt{s(r-1)+\frac{nkr^2}{m}(1-\frac{k}{m})}, \tag{20}$$

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

## VI. Counter Tree Based Maximum Likelihood Estimation

In this section, we propose another flow-size estimator called Counter Tree based Maximum likelihood Estimation (CTM).

### A. CTM Method

From (11), the probability of $Z_i=z_i$ is

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

The value of $n$ is known from the Counter Tree. The values of $m$ and $k$ are determined by prescribed system parameters $M$, $b$ and $d$, $h$. Hence, $P\{Z_i=z_i\}$ can be written as a function of $z_i$, denoted as $p(z_i)$. Therefore, the probability for observing $X_i=x_i$ can be calculated by

$$\begin{aligned}Prob\{X_i=x_\}&=Prob\{Y_i+Z_i=x_i\}\\&=\sum_{z_i=0}^{x_i}p(z_i)Prob\{Y_i=x_i-z_\}\\&=\sum_{z_i=0}^{x_i}p(z_i)q(s,y_i),\end{aligned} \tag{21}$$

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

$$L(s;x_0,x_1,\ldots,x_{r-1})=\prod_{i=0}^{r-1}\sum_{z_i=0}^{x_i}p(z_i)q(s,y_i). \tag{22}$$

Therefore, we have

$$\frac{d\ln L}{ds}=\sum_{i=0}^{r-1}\frac{\sum_{z_i=0}^{x_i}p(z_i)q(s,y_i)(\sum_{j=0}^{y_i-1}\frac{1}{s-j}+\ln(1-\frac{1}{r}))}{\sum_{z_i=0}^{x_i}p(z_i)q(s,y_i)}. \tag{23}$$

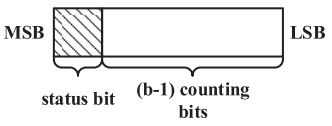



Fig. 6. A $b$-bit counter consists of $(b-1)$ counting bits and a status bit.

The derivation process is given in Appendix A. The value of $s$ that satisfies $\frac{d\ln L}{ds}=0$ will maximize $\ln L$ and thereby the likelihood function $L$. In addition, we observe that $\frac{d\ln L}{ds}$ is monotonically decreasing with respect to $s$. Therefore, the bisection search method can be used to find the $s$ value such that $\frac{d\ln L}{ds}=0$. As a result, we obtain an estimator for $s$ as follows:

$$\hat{s}=\arg\max_{s}\{\ln L\}=\{s|\frac{d\ln L}{ds}=0\}. \tag{24}$$

### B. Analysis of $\hat{s}$

Due to space limitation, the analysis of $\hat{s}$ is given in Appendix B. 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 $\alpha$ confidence interval for $s$ is

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

## VII. Enhanced Counter Tree Architecture

### A. Motivation

Recall that the design of vertical counter sharing in the Counter Tree reserves counters at higher layers for large flows. However, the question that which flows have actually used the high-layer counters is left open since each high-layer counter can be shared by multiple virtual counters. If two flows share a high-layer counter and only one of them uses the counter for recording its packets, there is no way to figure out which of the two actually uses that counter from the values in the Counter Tree, which leads to ambiguity. Consider a simple example in which $f$ is a small flow with size 1 and $g$ is large flow with size 30,000. As shown in Fig. 3, we assume the packet of $f$ is recorded in $V[1]$ (more specifically $C[1]$ since other counters in $V[1]$ are not used), and 10,000 of $g$'s packets are recorded in $V[3]$. As a result, $C[12]$ is used by $g$ to accommodate such a large number of packets. We know that $V[1]$ and $V[3]$ share the component counter $C[12]$. Although $C[12]$ has never been used by $f$, it is also included as a component in $f$'s virtual counter $V[1]$. As a result, a great noise is introduced by $g$ when we estimate the size of $f$ since $C[12]$ is included in the estimation of $f$. To handle this problem, our previous design relies on mapping each flow to many virtual counters (which helps amortizing the noise) and using statistical means to remove the noise. However, we suspect that large noise introduced at higher layers can nevertheless degrade the estimation accuracy. In this section, we introduce additional mechanism to address this issue.

We define the *length* of each virtual counter as the number of component counters it truly uses. In the previous example, the length $V[1]$ is 1 since only $C[1]$ is used, while the length of $V[3]$ is 3 since $C[3]$, $C[9]$ and $C[12]$ have been used. We observe that the aforementioned ambiguity results from the uncertainly of the length of each virtual counter after recording. In the previous design of Counter Tree, we simply assume every virtual counter has the same length, which is equal to the height $h$ of the tree. The experiment results in Section VIII will demonstrate that Counter Tree works well when the tree height $h$ is set small, e.g., $h=2$. However, the performance of Counter Tree seriously deteriorates when the tree height grows for the purpose of extending estimation range. Theoretically, this observation is embodied by the fact that the variance of $\hat{s}$ increases exponentially with $h$ according to (17). To accommodate large flows, reasonably large $h$ should be used, leading to large estimation error. Therefore, we must figure out how to resolve the ambiguity.

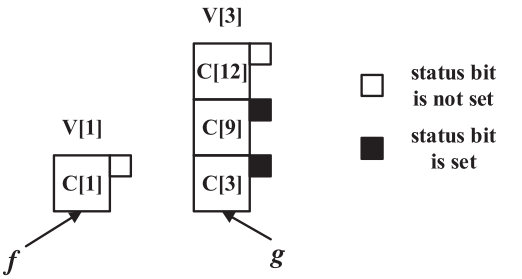



Fig. 7. After introducing status bits, we can calculate the lengths of virtual counters based on the status bits in their component counters. The length of $V[1]$ that $f$ is mapped to is 1 since the status bit of $C[1]$ is not set, and the length of $V[3]$ that $g$ is mapped to is 3 since the status bits of $C[3]$ and $C[9]$ are set while the status bit of $C[12]$ is not set.

### B. Counters With Status Bits

To address this issue, we propose an Enhanced Counter Tree (ECT) architecture which determines the length of each virtual counter and thereby resolves the ambiguity by adding a status bit to each counter. For each $b$-bit counter, its lower $(b-1)$ bits are used for counting packets, and its most significant bit is used as the *status bit*. See Fig. 6 for illustration. Only if the counter overflows, will the status bit of that counter be set. With the status bits, we can calculate the true length for each individual virtual counter, thereby reducing the noise caused by vertical counter sharing among virtual counters. More specifically, to build a virtual counter according to its length, we traverse along the path from its layer-0 counter to the root, and include all counters until the first counter (included) whose status bits has not been set. Following the aforementioned example, Fig. 7 shows that the introduction of status bits can alleviate the ambiguity in sharing at higher layers. In the figure, $C[1]$ is a component counter of $V[1]$ that $f$ is mapped to. Since $C[1]$ does not overflow (i.e., its status bit is not set), the length of $V[1]$ must be 1 and therefore the component counter $C[12]$ should has been included. In contrast, the length of $V[3]$ (which $g$ is mapped to) is 3 since the status bits of both $C[3]$ and $C[9]$ are set while the status bit of $C[12]$ is not. In conclusion, the extra status bits can provide better resolution for virtual counters and help resolve ambiguity about whether a high-layer component counter should be included in a virtual counter or not.

If a counter overflows due to a large flow, the small flows that share the counter will observe a large counter value. However, each flow is assigned to $r$ counters at the bottom layer, where $r$ is large, e.g., in hundreds. The number of large flows is much smaller than the number of small/medium flows in real traffic. For a small flow, if a few of its counters are overflowed due to sharing with large flows, the values of these counters will be large. But as long as most of the small flow's counters have small values, the effect of counter crosstalk will be dampened by the maximum likelihood estimation (24), which discounts the outliers (large counter values).

For the first estimator (13), when there are many, many flows, even without a large flow, the sheer number of small flows mapped to a counter may cause the counter to overflow. But this noise can be statistically measured and removed when $r$ is sufficiently large. Among the $r$ counters, statistically, some will carry larger-than-average noise and some will carry smaller-than-average noise. When $r$ is large enough, such difference will be evened out due to the law of large numbers. The average noise term is captured by the second term in (13).

### C. Recording and Estimation

After employing status bits, the process of online packet recording remains the same except that we need to set its status bit when a counter overflows. For the offline estimation process, we should slightly modify the estimators given in (13) and (24) to reflect the change. Consider the estimation of flow $f$. Let $l_i$ be the length of the virtual counter $V_f[i]$ after the measurement period, where $0 \leq i < r$. Instead of using a unified height $h$ for all subtree counters, the height of the subtree counter corresponding to $V_f[i]$ should be $l_i$, and the value $k$, meaning the number of leaf nodes in the subtree, is therefore $d^{l_i-1}$. For example, in Fig. 7, the height of the subtree counter corresponding to $V[1]$ is 1 and it contains only one leaf node, while the height of the subtree counter corresponding to $V[3]$ is 3 and it contains $2^{3-1}$ leaf nodes. Everything else remains the same for the offline estimation. Our experiments in Section VIII will show that the ECT with status bits remarkably improves the estimation accuracy of CT.

## VIII. Experiments

### A. Experiment Setup

We have implemented the two estimators based on the Counter Tree, i.e., CTE and CTM, from Section V and Section VI, respectively. CTE and CTM share the same module for online packet recording, which performs the operations described in Section IV. Hence, when we evaluate the online operations of the Counter Tree, we use CT as the abbreviation. We have also implemented the Enhanced Counter Tree architectures. We compare them with the most related counter architectures: (1) randomized counter sharing (MLM) [7], [8] and (2) Counter Braids (CB) [5], [6]. The bitmap-based MSCBF is less memory efficient than the counter architectures [7]. Hence, we do not include it for comparison. 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 University of Florida, and we are authorized to store the packets headers in disk for our experiments. The trace contains about 68 million TCP flows and 750 million packets. We implement those counter architectures in software and evaluate their performance by running experiments on the trace. Suppose each measurement period contains 10 million packets. We divide the trace into measurement periods, and perform different estimators in each period. We use all 750 million packets in our experiments, for 75 periods. We randomly pick the results from one period to report. In fact, the results from different periods are quite similar. During the chosen period, there are 1,070,632 TCP flows and 10,053,234 packets, and the minimum, average, and maximum flow size is 1,939, and 10,972 packets, respectively.

We conduct four sets of experiments. The first set is used for comparing the estimation accuracy and range of CT, MLM and CB. We vary the available memory space $M$ from 0.125MB, 0.25MB, 0.5MB, to 1MB, which translate to about 1bit/flow, 2bits/flow, 4bits/flow, and 8bits/flow, respectively. For Counter Braids, we use the same settings as in the original paper [5]: A two-layer CB and three hash functions at both layers. The layer-0 counters are 8 bits deep and the layer-1 counters are 56 bits deep. For MLM, we set the counter length to 6 bits and the size of each storage vector to 100 as in [7]. For CTE and CTM, we implement a 2-layer tree (which is sufficient for our experiments) with $d = 2$ and $b = 4$ by default. For fair comparison with MLM, we set the size $r$ of each virtual counter array to 100. The second set of experiments compare the processing overhead of online packet recording of these counter architectures. In the third set of experiments, we will vary the values of $d$, $b$ and $r$ to study their respective impact on performance. In the fourth set of experiments, we compare the performance of Counter Tree and Enhanced Counter Tree under different tree heights.

### B. Estimation Accuracy and Range

Recall that the main objective of the paper is to design a new counter architecture that can work in very tight space where existing counter architectures no long work well. So we first compare Counter Tree with CB and MLM in terms of estimation accuracy and range to see how they work under the same available memory. The estimation results of CB are shown in Fig. 8 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 $M = 0.125$MB is used, CB cannot produce any meaningful results. When $M = 0.5$MB, CB generates positively biased results that are all above the equality line. When the available memory space increases to 1MB, the estimation results of CB do not converge. So we terminate the process after 1000 iterations. We find that when $M \geq 2$MB, CB can yield very accurate estimates (which is not shown in the figure). Therefore, CB does not suite for traffic measurement under very tight memory.

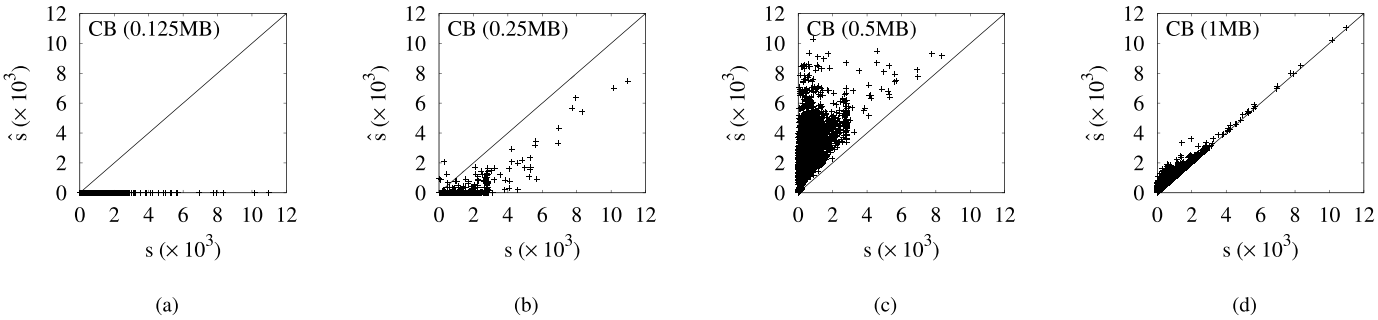



Fig. 8. Fig. 8a shows estimation results by Counter Braids when $M = 0.125$MB. Each point in the plot represents an $(s, \hat{s})$ pair for a particular flow, where the $x$ coordinate is the true flow size $s$ and the $y$ coordinate is the estimated flow size $\hat{s}$. The equality line, $y = x$, is presented for reference: A point closer to the equality line is more accurate. Fig. 8b shows estimation results by Counter Braids when $M = 0.25$MB. Fig. 8c shows estimation results by Counter Braids when $M = 0.5$MB. Fig. 8d shows estimation results by Counter Braids when $M = 1$MB.

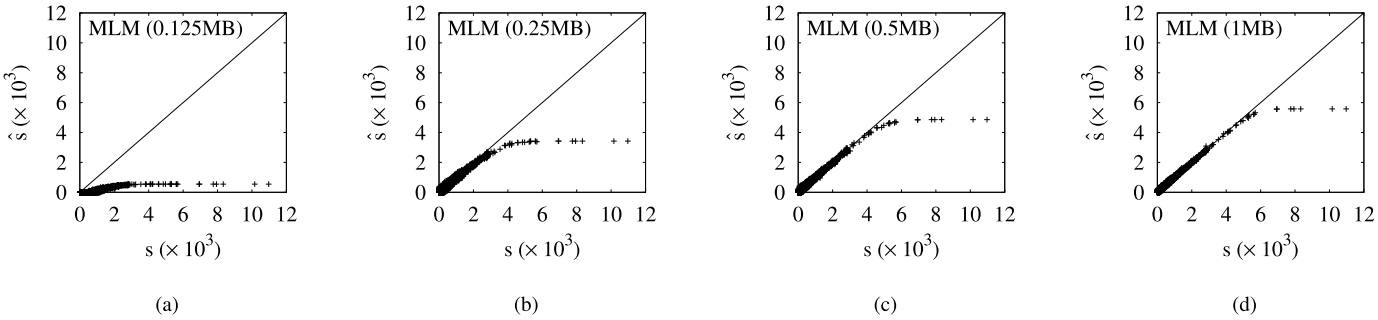



Fig. 9. (a)–(d) Estimation results by MLM when $M = 0.125$MB, 0.25MB, 0.5MB, and 1MB, respectively.

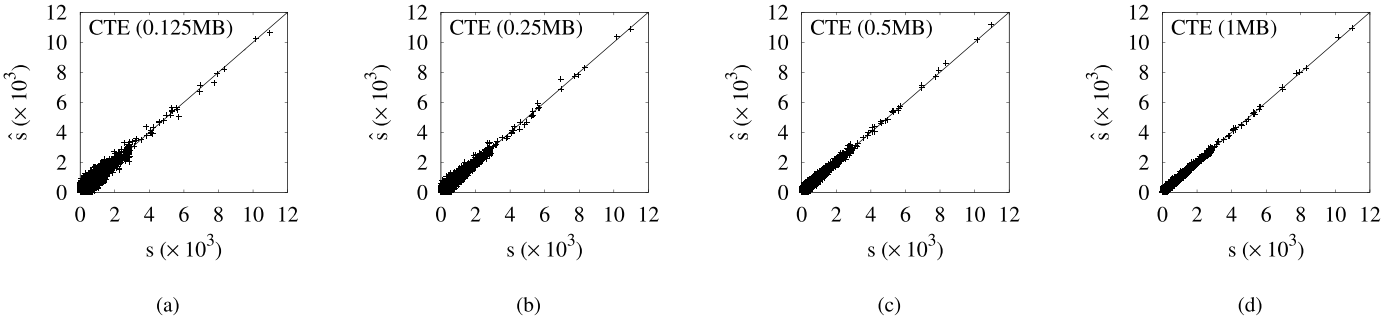



Fig. 10. (a)–(d) Estimation results by CTE when $M = 0.125$MB, 0.25MB, 0.5MB, and 1MB, respectively.

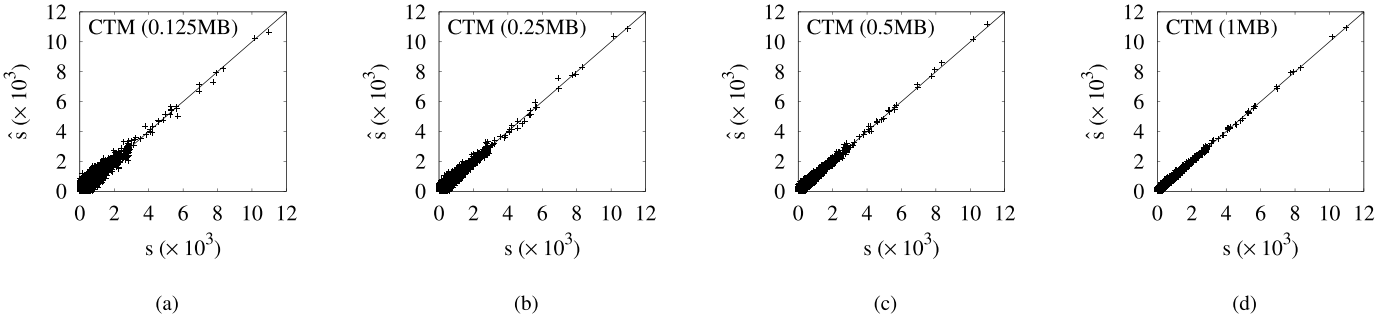



Fig. 11. (a)–(d) Estimation results by CTM when $M = 0.125$MB, 0.25MB, 0.5MB, and 1MB, respectively.

Fig. 9 presents the estimation results of MLM. MLM does not work when $M = 0.125$MB. Although MLM can yield accurate estimates for small or moderate flows when more memory is available, it produces very negatively biased results for large flows. Because large flows may lead to counter overflows, some packets cannot be recorded when the counters are fully used. Although the increase of $M$ can enlarge the estimation range of MLM to some extent, it does not address the problem fundamentally. For example, the estimation range is about 6000 when $M = 1$MB.

The estimation results of CTE and CTM are given in Fig. 10 and Fig. 11, respectively. As expected, the employment of the Counter Tree architecture significantly extends the counting range than MTM. Both CTE and CTM can yield very accurate estimates for all flows, including flows with very large sizes, even under a tight memory. 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. 12.

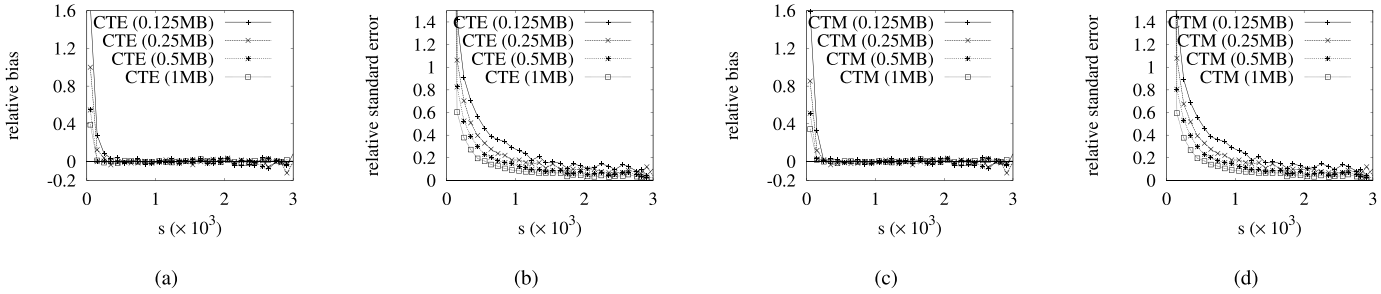



Fig. 12. Fig. 12a shows the relative estimation bias $Bias(\frac{\hat{s}}{s})$ of CTE. Fig. 12b shows the relative standard error $StdErr(\frac{\hat{s}}{s})$ of CTE. Fig. 12c shows the relative estimation bias $Bias(\frac{\hat{s}}{s})$ of CTM. Fig. 12d shows the relative standard error $StdErr(\frac{\hat{s}}{s})$ of CTM.

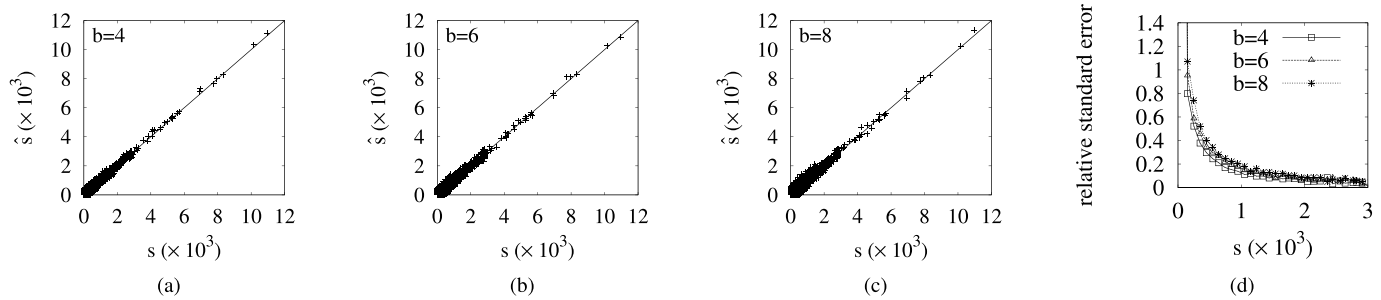



Fig. 13. Impact of $b$ on the performance of CTE, where $M = 0.5$MB, $d = 2$, and $r = 100$. Fig. 14a shows estimation results of CTE when $b = 4$. Fig. 14b shows estimation results of CTE when $b = 6$. Fig. 14c shows estimation results of CTE when $b = 8$. Fig. 14d shows the comparison of relative standard error when $b$ = 4, 6, 8.

We find that CTE and CTM in fact have comparable estimation accuracy. Fig. 12 shows that the relative errors for large flows are small. Generally, $Bias(\frac{\hat{s}}{s})$ and $\frac{\sqrt{Var(\hat{s})}}{s}$ decrease with the increase of $s$. Although the relative errors for small flows can be large, the results in Fig. 10 and Fig. 11 demonstrate that no small flows will deviate significantly from the equality line for large absolute errors (which would cause mis-classification). It is expected and true that the relative errors for small flows are large for virtually all estimation methods. We use an extreme example to bring out the idea: There is a difference in interpreting the results for small flows and those for large flows. Consider a small flow of size 1. If the estimation is 2 (which is in fact a good result because it is off by just 1), the relative error is 100%. For a large flow of 10,000, if the relative error is 100%, it will be 20,000, a truly bad estimation. For the same large flow, if the estimation is off by 10, it is a great estimation because the relative error is just 0.1%. However, if the previous small flow is off by 10, the relative error is 1000% — even with such a relative error, we would not necessarily say this is a bad estimation because a flow of 11 packets will not be mis-classified as a large flow, in applications of identifying elephant flows or classifying all flows into a few categories based on their sizes.

To numerically evaluate how large flows affect the estimation results of small flows, we only consider small flows that share counters with large flows, and omit the small flows that do not share any counters with large flows. More specifically, we consider all flows with sizes smaller than 100 as small, while those with sizes larger than 1000 as large. However, we observe that all small flows share multiple counters with large flows. The reason is that each flow uses a large number $r$ of counters, which means a small flow will have a good chance to share at least one counter with one of the large flows. We stress that our method is designed to handle such sharing through noise removal or maximum likelihood estimation.

In conclusion, CT works much better than CB and MLM under a tight memory, and it significantly extends the estimation range when compared with MLM.

### C. Processing Overhead of Online Packet Recording

The processing overhead for recording a packet is mainly due to memory accesses to read and write counters and the computations of hash values. A typical implementation of Counter Braids requires 3 hash functions at each layer, mapping each flow to the corresponding counters. To record a packet, the router needs to read the 3 counters at 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 at the second layer, which requires 3 more hash computations. Hence, the lower bounds on the number of memory accesses and the number of hash computations by CB are 6 and 3, respectively. In contrast, MLM aligns all counters at 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 recording 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 improve over CB by a factor of about 3. achieve 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.

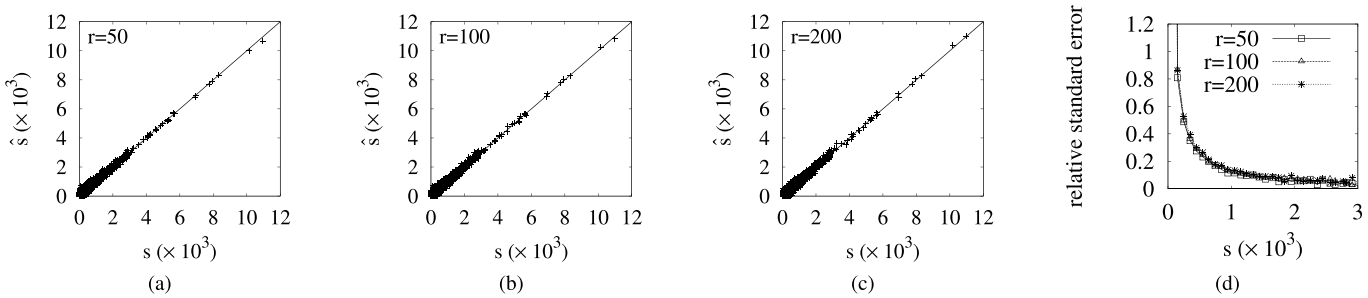



Fig. 14. Impact of $r$ on the performance of CTE, where $M = 0.5$MB, $b = 4$ and $d = 2$. (a)–(c) Estimation results by CTE when $r$ = 50, 100, and 200, respectively. (d) The comparison of relative standard error when $r$ = 50, 100, 200.

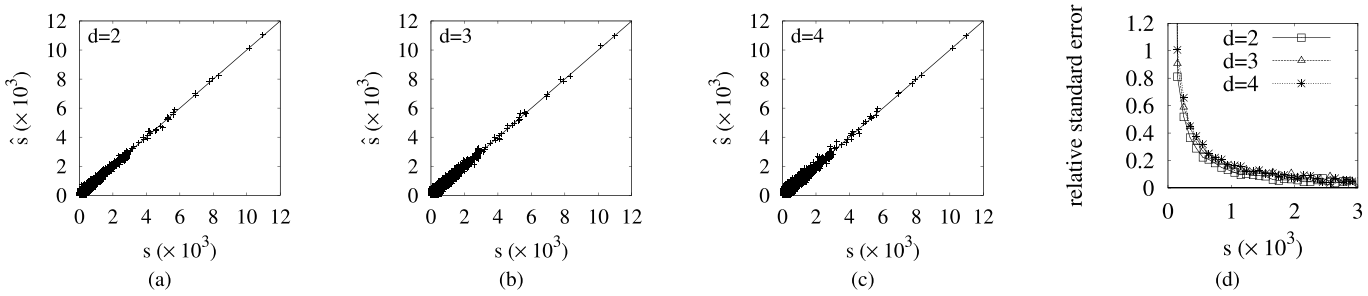



Fig. 15. Impact of $d$ on the performance of CTE, where $M = 0.5$MB, $b = 4$ and $r = 100$. (a)–(c) Estimation results by CTE when $d$ = 2, 3, and 4, respectively. (d) The comparison of relative standard error when $d$ = 2, 3, 4.

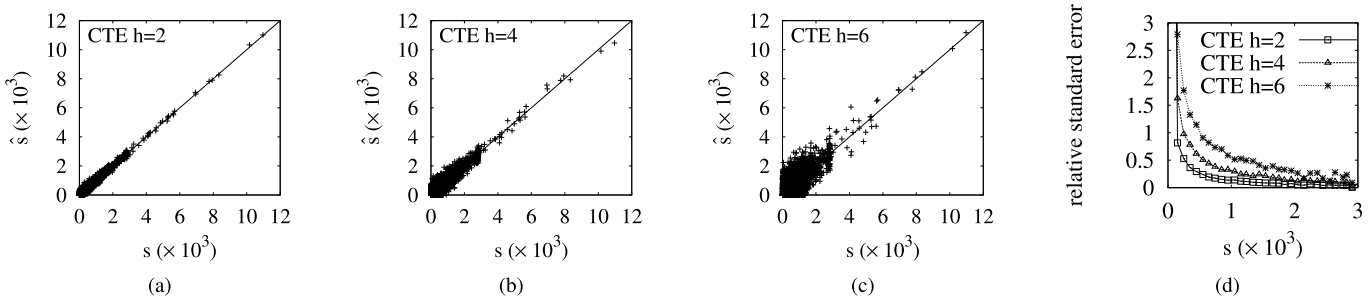



Fig. 16. Performance of CTE with different tree height $h$, where $M = 0.5$MB, $b = 4$, $d = 2$ and $r = 100$. (a)–(c) Estimation results by CTE when $h$ = 2, 4, and 6, respectively. (d) The comparison of relative standard error when $h$ = 2, 4, 6.

TABLE I

COMPARISON OF AVERAGE PROCESSING OVERHEAD FOR RECORDING A PACKET BY CB, MLM AND CT

| memory size (MB) | number of memory accesses | | | number of hash computations | | |
|---|---|---|---|---|---|---|
| | CB | MLM | CT | CB | MLM | CT |
| 0.125 | 6.02 | 2 | 2.11 | 3.01 | 1 | 1 |
| 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 |

### *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 $d = 2$, $r = 100$, $h = 2$ and vary $b$ from 4, 6 to 8.
2) Impact of $r$: we fix $b = 4$, $d = 2$, $h = 2$ and vary $r$ from 50, 100 to 200.
3) Impact of $d$: we fix $b = 4$, $r = 100$, $h = 2$ 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. 13, Fig. 14, and Fig. 15. When we increase the $b$, $r$ or $d$, the estimation range will be increased accordingly. But this does not come for free. Since the increase of $b$, $r$ or $d$ makes more flows share each counter, it is expected that the estimation accuracy will degrade due to elevated noise. However, the experimental results show that such degradation is small, and the performance of CTE is not very sensitive to the change of $b$, $r$ or $d$ (We will show shortly that the change of $h$ can significantly affect the estimation accuracy of CTE).

### *E. Comparison of CTE and E-CTE*

Recall that our analysis in Section VII points out that the increase of tree height $h$ can degrade the performance of CTE and CTM. To demonstrate this, we run experiments using the CTE method (similar results can be observed if the CTM method is adopted) with different tree heights. We fix $M = 0.5$MB, $b = 4$, $d = 2$, $r = 100$, and vary $h$ from 2, 4, to 6. The results are presented in Fig. 16. As we expect, the estimation accuracy of CTE becomes much worse with the increase of $h$. The fourth plot in Fig. 16 demonstrates that a larger $h$ significantly increases the relative standard error of the estimates. We use E-CTE to stand for the CTE method under the Enhanced Counter Tree architecture which is designed to address this issue. For comparison, we conduct the same experiments on E-CTE, and the results are depicted in Fig. 17. Owing to the status bits in E-CTE, the increase of $h$ only slightly degrades the performance of ECT. Therefore,

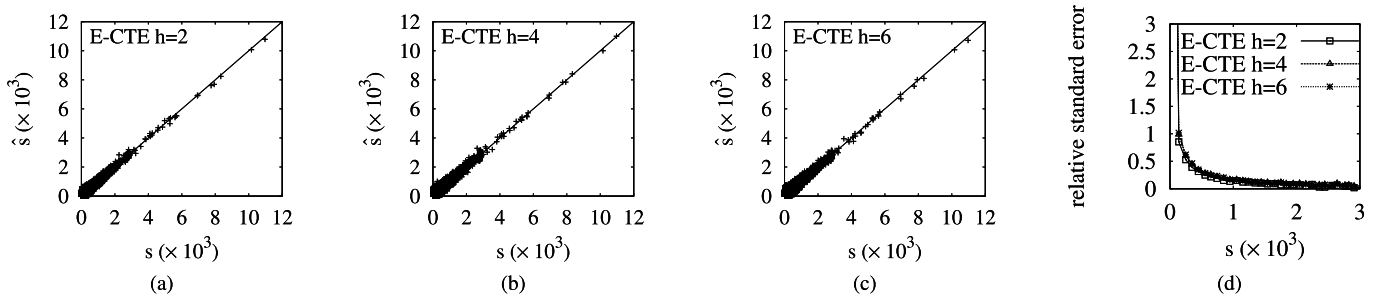



Fig. 17. Performance of E-CTE with different tree height $h$, where $M = 0.5$MB, $b = 4$, $d = 2$ and $r = 100$.

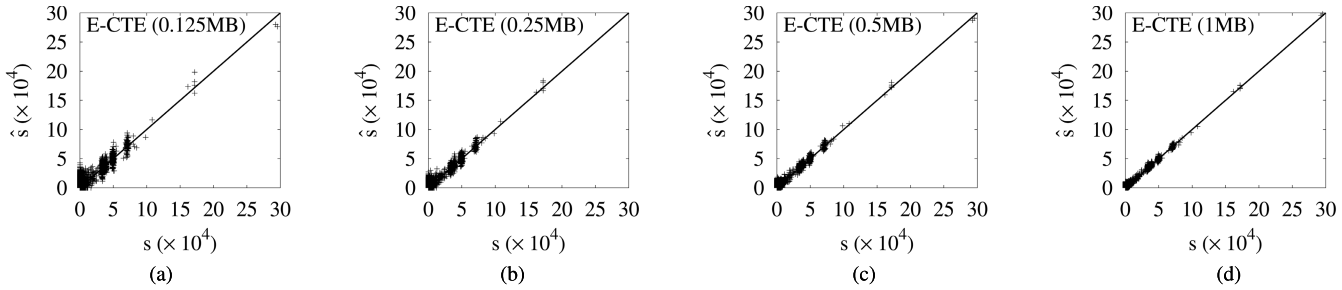



Fig. 18. Estimation results of E-CTE for a trace with 100 million packets. We set $b = 4$, $d = 2$, $r = 100$, $h = 4$, and (a)–(d) $M = 0.125$MB, 0.25MB, 0.5MB, and 1MB, respectively.

ECT dramatically outperforms CTE when a relatively large $h$ is adopted, e.g., $h = 6$.

### *F. Scalability of Counter Tree*

To evaluate the scalability of Counter Tree, we apply it to a larger trace, which contains 8,253,368 TCP flows and 100,000,015 packets, and the minimum, average, and maximum flow size is 1, 12.12 and 295,451 packets, respectively. The available memory space $M$ remains to be 0.125MB, 0.25MB, 0.5MB, and 1MB, which translate to about 0.125bits/flow, 0.25bits/flow, 0.5bits/flow, and 1bits/flow, respectively. We implement Counter Tree with different system parameters. Due to space limitation, Fig. 18 only shows the estimation results of E-CTE with $b = 4$, $d = 2$, $r = 100$, and $h = 4$ (similar results are observed under other settings). It is clear that Counter Tree can still yield reasonably accurate estimation results under an extremely tight memory space, e.g., 0.125 bits/flow.

## IX. Related Work

The Counter Braids [5], [6], the randomized counter sharing scheme [7], and our Counter Tree architecture improve the memory efficiency for per-flow traffic measurement by enabling statistical sharing among physical counters. Some prior work takes a different design path to save memory consumption of each counter (with the same counting range), thereby improving the memory efficiency. Small Active Counters (SAC) [27] divides each counter into two parts: an estimation part $A$ and an exponent part $B$. SAC derives an estimator $\hat{s} = A2^{qB}$, where $q$ is a global parameter. In [28], a memory-efficient method called DIScount COunting (DISCO) is proposed for measuring the flow size or flow byte. DISCO regulates the counter value to be a real increasing concave function of the actual flow size or flow byte, such that the counter value increases more slowly than the number of encoded flow packets or flow bytes. Note that both SAC and DISCO can be applied to those sharing counter architectures, including Counter Tree, to further improve memory efficiency.

Some prior art studies 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 [12] 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 [13] employs an array of bitmaps, each having a different sampling probability, to extend estimation range. Probabilistic Counting with Stochastic Averaging (PCSA) [14] (also known as FM sketch) maps each element to the $i$th (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 [15] and HyperLogLog [16] 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 significantly improve the estimation accuracy of PCSA. In [29], a series of improvements to HyperLogLog are proposed to further reduce its memory requirements and increase its accuracy for an important range of cardinalities.

## X. Conclusion

In this paper, we design a scalable counter architecture called 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 record a packet, Counter Tree only requires a little more than 2 memory accesses on average. We propose two offline methods to estimate flow sizes. Moreover, we propose the Enhanced Counter Tree architecture by adding a status bit for each counter to further improve the performance. The extensive experiments with real network trace demonstrate that our methods can yield good estimates even under very tight memory space.

## REFERENCES


[1] D. Shah, S. Iyer, B. Prabhakar, and N. McKeown, "Analysis of a statistics counter architecture," in *Proc. Hot Interconnects*, Aug. 2001, pp. 107–111.

[2] S. Ramabhadran and G. Varghese, "Efficient implementation of a statistics counter architecture," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 1, pp. 261–271, Jun. 2003.

[3] Q. Zhao, J. Xu, and Z. Liu, "Design of a novel statistics counter architecture with optimal space and time efficiency," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 34, no. 1, pp. 323–334, Jun. 2006.

[4] A. Kumar, J. Xu, and J. Wang, "Space-code bloom filter for efficient per-flow traffic measurement," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 12, pp. 2327–2339, Dec. 2006.

[5] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter braids: A novel counter architecture for per-flow measurement," in *Proc. ACM SIGMETRICS*, Jun. 2008, pp. 121–132.

[6] Y. Lu and B. Prabhakar, "Robust counting via counter braids: An error-resilient network measurement architecture," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 522–530.

[7] T. Li, S. Chen, and Y. Ling, "Fast and compact per-flow traffic measurement through randomized counter sharing," in *Proc. IEEE INFOCOM*, Apr. 2011, pp. 1799–1807.

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

[9] M. Chen and S. Chen, "Counter tree: A scalable counter architecture for per-flow traffic measurement," in *Proc. IEEE ICNP*, Nov. 2015, pp. 111–122.

[10] Y. Zhou, Y. Zhou, M. Chen, Q. Xiao, and S. Chen, "Highly compact virtual counters for per-flow traffic measurement through register sharing," in *Proc. IEEE GLOBECOM*, Dec. 2016.

[11] S. Chen, M. Chen, and Q. Xiao, *Traffic Measurement for Big Network Data,* "Wireless Networks." Cham, Switzerland: Springer, 2016.

[12] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Trans. Database Syst.*, vol. 15, no. 2, pp. 208–229, Jun. 1990.

[13] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high-speed links," *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, pp. 925–937, Oct. 2006.

[14] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for database applications," *J. Comput. Syst. Sci.*, vol. 31, pp. 182–209, Sep. 1985.

[15] M. Durand and P. Flajolet, "Loglog counting of large cardinalities," in *Proc. Eur. Symp. Algorithms*, 2003, pp. 605–617.

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

[17] *Access Time*, accessed on Nov. 5, 2016. [Online]. Available: http://www.webopedia.com/TERM/A/access_time.html

[18] *NEC and Corning Achieve Petabit Optical Transmission*, accessed on Nov. 5, 2016. [Online]. Available: http://optics.org/news/4/1/29

[19] N. Duffield, C. Lund, and M. Thorup, "Estimating flow distributions from sampled flow statistics," in *Proc. ACM SIGCOMM*, Oct. 2003, pp. 325–336.

[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*, Jun. 2004, pp. 177–188.

[21] N. Kamiyama and T. Mori, "Simple and accurate identification of high-rate flows by packet sampling," in *Proc. IEEE INFOCOM*, Apr. 2006, pp. 1–13.

[22] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 1, p. 75, Jan. 2002.

[23] O. Rottenstreich and I. Keslassy, "The bloom paradox: When not to use a bloom filter," *IEEE/ACM Trans. Netw.*, vol. 23, no. 3, pp. 703–716, Jun. 2015.

[24] G. Einziger and R. Friedman, "Counting with tinytable: Every bit counts!" in *Proc. IEEE INFOCOM Workshops*, Apr. 2015, pp. 77–78.

[25] J. Mikians, A. Dhamdhere, C. Dovrolis, p. Barlet-Ros, and J. Solé-Pareta, "Towards a statistical characterization of the interdomain traffic matrix," in *Proc. NETWORKING*, 2012, pp. 111–123.

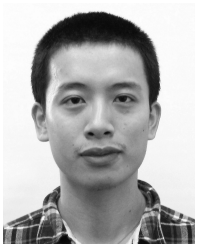
[26] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The count-min sketch and its applications," *J. Algorithms*, vol. 55, no. 1, pp. 58–75, Apr. 2005.

[27] R. Stanojevic, "Small active counters," in *Proc. IEEE INFOCOM*, May 2007, pp. 2153–2161.

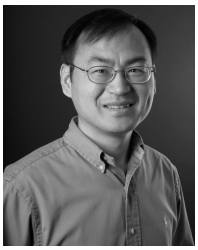
[28] C. Hu *et al.*, "Discount counting for fast flow statistics on flow size and flow volume," *IEEE/ACM Trans. Netw.*, vol. 22, no. 3, pp. 970–981, Jun. 2014.

[29] S. Heule, M. Nunkesser, and A. Hall, "HyperLogLog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm," in *Proc. ACM EDBT*, Mar. 2013, pp. 683–692.

[30] E. Lehmann, G. Casella, *Theory of Estimation*. New York, NY, USA: Springer-Verlag, 1998

**Min Chen** received the B.E. degree in information security from the University of Science and Technology of China in 2011 and the M.S. and Ph.D. degrees in computer science from the University of Florida in 2015 and 2016, respectively. His advisor is Dr. S. Chen. His research interests include Internet of Things, big network data, next-generation RFID systems, and network security.



**Shigang Chen** (A'03–M'04–SM'12–F'16) received the B.S. degree in computer science from the University of Science and Technology of China in 1993, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana–Champaign in 1996 and 1999, respectively. He was with Cisco Systems for three years. He joined the University of Florida in 2002. He served on the Technical Advisory Board for Protego Networks from 2002 to 2003. He is currently a Professor with the Department of Computer and Information Science and Engineering, University of Florida. He has authored over 100 peer-reviewed journal/conference papers. He holds 11 U.S. patents. His research interests include computer networks, internet security, wireless communications, and distributed computing. He received the IEEE Communications Society Best Tutorial Paper Award in 1999 and the NSF CAREER Award in 2007. He served in the Steering Committee of the IEEE IWQoS from 2010 to 2013. He is an Associate Editor of the IEEE/ACM TRANSACTIONS ON NETWORKING, *Journal of Computer Networks*, and the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY.

**Zhiping Cai** (M'08) received the B.S., M.Sc., and Ph.D. degrees in computer science from the National University of Defense Technology, Changsha, China, in 1996, 2002, and 2005, respectively. He is currently an Associate Professor with the School of Computer, National University of Defense Technology. His current research interests include network security and network virtualization.