Cache Assisted Randomized Sharing Counters in Network Measurement

Qian Liu
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, Jiangsu, CHINA
liu6qian0@163.com

Haipeng Dai
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, Jiangsu, CHINA
haipengdai@nju.edu.cn

Alex X. Liu
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, Jiangsu, CHINA
Department of Computer Science and Engineering
Michigan State University
East Lansing, MI, USA
alexliuxy@yahoo.com

Qi Li
Air Force Engineering University
Xian, Shanxi, CHINA
xiaomi9nyxs@126.com

Xiaoyu Wang
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, Jiangsu, CHINA
xiaoyuwang@smail.nju.edu.cn

Jiaqi Zheng
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, Jiangsu, CHINA
jiaqi369@gmail.com

ABSTRACT

This paper proposes a new counter architecture for network measurement called Cache Assisted and randomizEd ShAring counteRs (CAESAR). One of the greatest challenges for per-flow traffic measurement is designing an online measurement module to keep up with the rapid growth of link speed. To address this challenge, we use a fast on-chip memory as the cache before the slow off-chip SRAM counters, thereby decreasing the accesses per flow to off-chip counters to improve time efficiency without any packet loss. We use randomized sharing counters among multiple flows in SRAM to achieve a compact data structure with high storage efficiency. By removing the impact from other flows sharing counters with a specific flow, we theoretically analyze the expectation and confidence interval of its estimated flow size accurately. In this paper, we use the real-world network traces for software simulations and FPGA experiments on the Xilinx Virtex-7 FPGA chip to validate our theoretical findings. The results show that CAESAR is up to 92.4% and 90% faster than prior work CASE and RCS respectively, and CAESAR reduces the average relative error of CASE and RCS by more than half.

CCS CONCEPTS

• **Networks** → *Network resources allocation*;

ICPP 2018, August 13–16, 2018, Eugene, OR, USA

ACM ISBN 978-1-4503-6510-9/18/08…$15.00
https://doi.org/10.1145/3225058.3225078

KEYWORDS

network measurement, cache, shared-counters

ACM Reference Format:
Qian Liu, Haipeng Dai, Alex X. Liu, Qi Li, Xiaoyu Wang, and Jiaqi Zheng. 2018. Cache Assisted Randomized Sharing Counters in Network Measurement. In *ICPP 2018: 47th International Conference on Parallel Processing, August 13–16, 2018, Eugene, OR, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3225058.3225078

1 INTRODUCTION

1.1 Motivation

Per-flow traffic measurement for network applications such as caching, intrusion detection, and scheduling has attracted more and more attention in recent years [2–5, 23, 32–39, 42]. It provides network fine-grained statistics that help display network access patterns, geographic/demographic traffic distribution among users, scanning speeds of worm-infected hosts, *etc.* [12–14, 20]. Generally, per-flow traffic measurement is designed to monitor hundreds of millions of flows (including elephant and mice flows [20, 24]) by using large amounts of counters which are updated at each incoming packet at line speed [20, 21, 42]. On the one hand, this is challenging for fast packets capturing, with the increase of Internet link speed (*e.g.*, OC-768 of 40 Gbps and OC-3072 of 160 Gbps) in large data center and campus/enterprise network scenarios. On the other hand, the average access time of slow DRAM is 40 ns, while that of expensive SRAM (*e.g.*, QDRII+SRAM) is 3-10 ns, neither is fast enough to keep up with packet arrival speed [24]. To overcome these dilemmas, the architecture synthesizes on-chip fast memory with just 1 ns for once access [24] as cache and normal off-chip SRAM counters, then it exploits the fast on-chip cache memory to achieve real-time traffic measurement without packet loss. What's more, in the absence of any compression between actual and stored values, it can achieve more accurate counting.

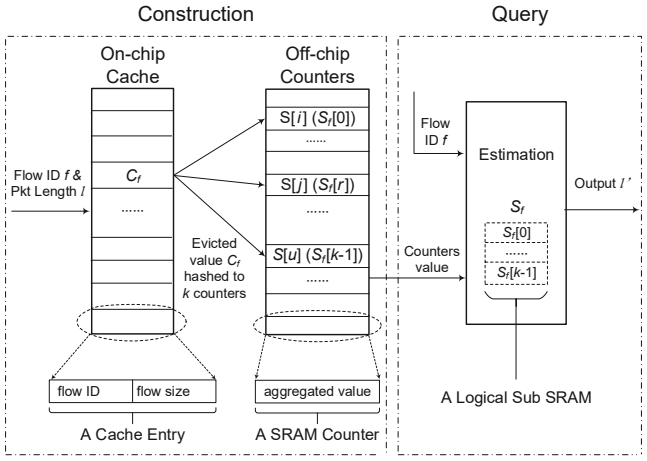


Figure 1: CAESAR architecture

1.2 Limitations of Prior Art

Without assistance of caches, there exists some schemes for flow size estimation such as SAC [29], ANLS [13], DISCO [12], CEDAR [30], and ICE-buckets [8]. They all rely on a single counter to record the specific flow using the compression method. Generally, they compress the large flow size to smaller stored one, and then use the corresponding decompression method to estimate the actual values. In these works, the compression function has high computational complexity and low storage efficiency. Another type of scheme is to use multiple counters to record each flow, such as Counter Braids [21, 25, 26], Randomized Counter Sharing (RCS) [20], First Loss Measurement Estimation (FLOE) [21], and Virtual HyperLogLog Counter (VHC) [41]. But slow access to off-chip SRAM counters can be very time consuming, leading to severe packet loss.

Sampling schemes are widely used in many off-the-shelf researches [6, 7, 9, 11, 15, 16, 18]. They designed different probabilistic sampling methods to screen out insignificant mice flows and identify meaningful elephant flows [1], all of which inevitably introduce estimation error due to filtered flows. Moreover, it is not only difficult to set an appropriate sampling rate, but they also maintain slow update rate of off-chip SRAM counters without fast on-chip memory.

Recently, the scheme Cache-Assisted Stretchable Estimator (CASE) [24] uses fast on-chip cache to help capture the flow packets, and then compresses each cache value to update specific mapped off-chip SRAM counters. However, its power operation in the compression phase adds to the computational complexity, and the one-to-one mapping between off-chip SRAM counters and flows reduces overall storage efficiency.

1.3 Proposed Approach

This paper proposes a new counter architecture called Cache Assisted and randomizEd ShAring counteRs (CAESAR) with high storage efficiency, high time efficiency, and high estimation accuracy. Our program has two phases, construction phase and query phase, as shown in Figure 1. The architecture involves two components. One is an assisted on-chip cache, a high-speed RAM which can catch up with packets seamlessly with high-speed links, and each entry stores a specific flow with its flow ID and flow size. The other is off-chip permanent SRAM counters, each counter stores the aggregated size of multiple flows.

In the construction phase, after capturing a new packet from the Internet, we generate a unique flow ID from its 5-tuple packet header. Then, we allocate an empty cache entry for a new flow or retrieve the occupied cache entry for the old flow which has been stored. We evict overflowed entries, as well as one entry if all the cache entries have been allocated. Afterwards, we uniformly divide the eviction value C_f into k parts and update them to k off-chip SRAM counters, mapped by k different collision-free hash functions.

In the query phase, we estimate the flow sizes with the off-chip SRAM counters. For a query flow ID f, we first select the k mapped counters using the above k predefined hash functions. Then, we remove the noise values added by other sharing flows from these k counters, and calculate the estimated size of flow f. Logically, the k mapped counters for a particular flow form a sub SRAM, *i.e.*, S_f as shown in Figure 1, which actually belongs to a unique off-chip SRAM.

1.4 Technical Challenges and Solutions

The first technical challenge is to estimate the evicted cache value of any flow. The cache eviction occurs in two unrelated cases that need to be considered separately: one is to overflow some cache entries, and the other is to exhaust all cache entries, obviously each flow may be evicted more than once. To address this challenge, we first presume that all the packets arrive at the same probability. For the first case, the empirical distribution of flow sizes shows that the majority of flows are below the average size, which means overflows seldom occur and even can be ignored with an appropriate cache entry capacity. For the second case, the chosen probability is independent of the cache value using an alternative algorithm such as Least Recently Used (LRU) or random replacement, which means any entry within the entry capacity can be evicted with equal probability. Consequently, for a specific flow, its multiple evictions can be formulated as a sequence of random integer variables that are independent and identically distributed, then we deduce the expectations and variances of its addition to its mapped SRAM counters. Similarly, we estimate the flow sizes of other flows sharing at least one counters with this specific one.

The second technical challenge is to estimate the noise for a given flow in its arbitrary mapped counter. To deal with the challenge, we first postulate that the flow size obeys a certain distribution. We hash k counters randomly and evenly for a specific flow, each of which may have noise added by other flows where the counters sharing probability can be calculated. Similar to the first challenge solution, we calculate the expectation of addition from each possible noise flow to its mapped counters. And then for an arbitrary mapped counter, we can calculate the aggregated noise from all possible noise flows according to the known flow size distribution.

The third technical challenge is to accurately de-noise the noise generated by other flows. Then it will be a trade-off between accuracy and computational complexity of de-noise calculation. In order to solve the problem, we try two widely-used de-noising methods,

moment estimation and maximum likelihood estimation, which are called CSM (Counter Sum estimation Method) and MLM (Maximum Likelihood estimation Method) in this paper. First, we prove our expectation of flow size is unbiased. Then, we respectively use these two different methods to estimate the specific Binomial distribution of estimations to a corresponding Gaussian distribution in CSM and in MLM respectively, and further deduce their variances to compare their evaluation accuracy. Finally, we theoretically prove that the estimations of CAESAR are more accurate than relatively accurate RCS, while CAESAR is much more flexible than RCS in off-chip memory size.

1.5 Advantages over Prior Art

We conduct both simulations and experiments to evaluate the performance of CAESAR and the related state-of-the-art schemes CASE and RCS using the real-world network traces. As the simulation, CASE hardly work because its estimation nearly mistakes all the flows to 0, resulting in approximately 100% relative error. On the contrary, CAESAR can accurately estimate flow sizes for most flows under the same cache size. Furthermore, the average relative errors for RCS estimation with empirical packet loss rates of $\frac{2}{3}$ and $\frac{9}{10}$ are 67.68% and 90.06%, respectively. In contrast, CSM and MLM estimations for CAESAR lead to average relative errors of 25.23% and 30.83%, respectively. As the experiments processing the same number of packets, we have implemented CAESAR as well as CASE and RCS on a Xilinx Virtex-7 FPGA chip with designed maximum clock rate of 18.912 MHz, supporting 680.832 Mbps throughput and 36-bit packet input bandwidth. Our experimental results show that CAESAR is on average 74.8% and up to 92.4% faster than CASE, and it is on average 75.5% and up to 90% faster than RCS.

1.6 Overview

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 describes the design of our CAESAR, which includes an online construction phase and an offline query phase. In Section 4 and Section 5, we conduct theoretical analysis about the two phases and deduce the estimation accuracy of our design. In Section 6, we conduct simulations and experiments to validate the performance of our proposed scheme in terms of accuracy and efficiency. At last, we conclude the paper in Section 7.

2 RELATED WORK

2.1 Compression Based Approach

There exist some schemes focus on using a single counter to record each flow, but none use an on-chip cache to help capture packets, and therefore fall short in packet loss when capturing wire-speed flows. First, some schemes like the SAC [29], ANLS [13], DISCO [12], CEDAR [30], and ICE-buckets [8] rely on compression methods. They respectively compress the actual flow sizes to smaller values in mapped counters, and then use corresponding decompression methods to recover. However, all these schemes require that the number of counters be at least equal to the quantity of recorded flows, and all counters have a uniform size for either elephant or mice flows, which inevitably leads to inefficient storage as significant bytes of counters for mice flows are wasted. Second, another kind of methods try to compress counter arrays by sharing counters among different flows. Counter Braids [21, 25, 26] uses a two-stage counter architecture, where three or more counters are allocated to a single flow. Nevertheless, each flow needs more than 4 bits, which leads to storage waste for mice flows with sizes of just 1 or 2 bits; and per-arrival packet updates at least three counters. To overcome the two limitations, Li *et al.* proposed Randomized Counter Sharing (RCS) for random data encoding/decoding [20], and First Loss Measurement Estimation (FLOE) which allocates less than 1 bit in counter per packet [21]. Moreover, the Virtual HyperLogLog Counter (VHC) proposed by Zhou *et al.* needs slightly more than 1 memory access per packet. Although they all reduce the memory access to improve the storage efficiency, the updating speed on off-chip SRAM is inherently too slow to meet the wire-speed processing of the real-world network traces, which further leads to severe packet loss.

2.2 Sampling Based Approach

There are some schemes using probabilistic sampling methods to filter out mice flows [1, 6, 11, 16] to enhance the performance of recording high-rate network flows. Based on the correlation between flow size and flow rate [40], Kamiyama *et al.* [15] proposed to identify high-rate flows through sampling packets with the timeout property. Moreover, Kodialam *et al.* improved the random sampling methods [9] by proposing a dual-run sampling method [17]. Many commercial routers (*e.g.*, Cisco's Net Flow) choose these sampling methods, but it is still hard to set appropriate sampling rates. Meanwhile, the filtered flows inevitably introduce significant estimation errors, and these schemes cannot keep up with high link speed due to the inherent speed limitation of off-chip SRAM counters without fast on-chip memory.

2.3 Cached Assisted Approach

Up to now, there is one scheme called Cache-Assisted Stretchable Estimator (CASE) [24] that records flows with the assistance of on-chip cache structure to achieve per-flow traffic measurement. First, it captures flow packets to count in a fast on-chip cache, and then compresses the recorded values and updates them to off-chip SRAM counters following one-to-one mappings. Nevertheless, CASE's allocation of counters is based on DISCO [12], and thus inevitably inherits its main drawback, *i.e.*, storage inefficiency. What's more, CASE needs time-consuming power operations in the compression step, which further makes CASE hard to achieve wire-speed processing of network flows. Compared with CASE, our CAESAR scheme is based on RCS [20] to improve storage efficiency, and CAESAR only involves a few calculations such as hash functions and additions, which consume less time.

3 DESIGN OF CAESAR

3.1 Online Construction Phase

As Figure 1 shows, our design consists of two main isolate memories, one is an on-chip cache directly linked to the network, the other is a permanent off-chip SRAM as the counters. The cache is usually much faster than regular SRAM. Our design uses a cache table with M cache entries, each of which includes a flow ID and flow size, and the size can be counted in either packets or bytes. When a new packet is captured, we first analyze its 5-tuple packet header, *i.e.*,

Table 1: Notations and symbols used throughout this paper

Symbol	Meaning
Q	Number of all flows
n	Number of packets of all flows
y	Maximum capacity of a cache entry
l	Maximum capacity of a SRAM counter
M	Number of entries in cache
L	Number of counters in SRAM
k	Number of mapped SRAM counters per flow
$\mathcal{P}_i$	Probability of flow size of i
μ	Expected flow size
x	Actual flow size of flow f
p, q	Two parameters satisfying $x = p \cdot k + q$ $(q < k)$
a, b	Two parameters satisfying $y = a \cdot k + b$ $(b < k)$
t	Times of eviction of flow f's cache entry
E_i	Value of flow f's i-th evicted item
p_i	Probability of cache evicting value of i
EV_i	Addition of E_i to an arbitrary counter among k flow f's mapped counters
EV_{i1}, EV_{i2}	Two parameters satisfying $EV_i = EV_{i1} \cdot k + EV_{i2}$ $(EV_{i2} < k)$
S_f	Logical sub SRAM counters for flow f
X	Value of an arbitrary chosen SRAM counter $S_f[r]$ $(0 \le r \le k-1)$ for flow f
Y	Portion of flow f added to X
z	Size of flow $\overline{f}$ other than flow f
z_1, z_2	Two parameters satisfying $z = z_1 \cdot k + z_2$ $(z_2 < k)$
Z	Portion of a flow $\overline{f}$ added to X
Z_{total}	Aggregated portion of all flows $\overline{f}$s other than flow f added to X
α	The reliability of a Gaussian distribution

source IP address, destination IP address, source port, destination port, and application protocol to generate a unique identifier flow ID. Then we try to retrieve this ID in the cache table with the identifier. If hit, we directly update its flow size (*i.e.*, add 1 to its packet count) or flow volume (*i.e.*, add the length of this packet to its byte count), or update them both. The experiments in Section 6 show that the flow size and flow volume have almost the same distribution, except for the magnitude, so we only focus on the flow size in the following analysis. Moreover, the recorded flow size may overflow due to the limited cache entry capacity, so we evict its value (*i.e.*, fulfilled cache entry) to off-chip SRAM. If miss, there are two possible situations. First, if the cache table is available, we allocate an empty entry for the new flow. Second, if the cache table has no empty entries, we use an alternative replacement algorithm to select an entry to be evicted, and we try both LRU and random replacement algorithms in this paper. Then we evict the cache data (*i.e.*, not fulfilled) to the off-chip SRAM, and allocate this entry to this new flow. Table 1 summarizes the notations used throughout this paper.

Suppose the quantity of off-chip SRAM counters is L. For each evicted value, namely, the recorded current flow size in the evicted

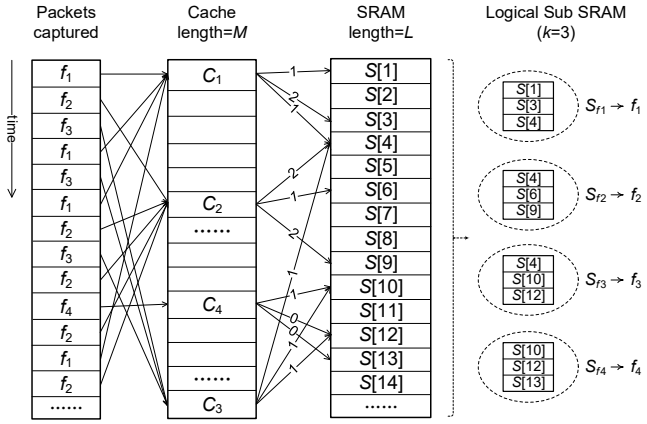


Figure 2: CAESAR updating process

entry, we divide it into k parts, and update them to randomly selected k counters in SRAM with k different collision-free hash functions. Note that these hash functions for mapping k counters are merely acting on the flow ID, and therefore, each flow is mapped to k fixed counters even if it is evicted more than once. At the end of the measurement, we dump all the cache entries to the SRAM counters.

Let C_f denote the corresponding cache entry for a flow, say f. Let $S[i]$s denote off-chip SRAM counters. We denote $S_f[r]$s $(r = 0, \ldots, k-1)$ as the mapped SRAM counters for flow f, as shown in Figure 1. Indeed, S_f is a logical subsequence of SRAM counter sequence, containing k SRAM counter $S[i]$s.

Suppose x is the actual size of flow f which is evicted only once after capturing all of its packets, and then suppose $x = p \cdot k + q$ $(p \in \mathbb{Z}^+ \bigcup \{0\}, q = 0, \ldots, k-1)$. If $q = 0$, x is divisible by k, then only an integer p is added to each of the k counter $S_f[r]$s $(r = 0, \ldots, k-1)$. Otherwise, if $q \neq 0$, we first equally divide the aliquot part, adding p to k mapped SRAM counters $S_f[r]$s respectively. Then we randomly allocate the rest q packets to these k counters one by one.

For the example shown in Figure 2, we have four flows, $f_1, \ldots, f_4$, recorded in the on-chip cache with evicted cache value $C_1 = 4$, $C_2 = 5$, $C_3 = 3$, and $C_4 = 1$, respectively. By using $k = 3$ hash functions, we get S_{f1} as the logical subsequence for flow f_1 which contains k counters, $S[1]$, $S[3]$, and $S[4]$ in Figure 2. Then, we first divide the aliquot part of C_1, namely 3, to k counters, and therefore, each counter gets an increment of $3/k$ (*i.e.*, $p = 1$). Next, we randomly allocate the rest part of C_1 (*i.e.*, $q = 1$) to these k counters (*e.g.*, $S[3]$ is chosen). Similarly, f_2 chooses $S[4]$, $S[6]$, and $S[9]$ as its S_{f2}, equally divides the aliquot part of C_2, *i.e.*, 3, and then randomly allocates the remainder, *i.e.*, 2, to the three counters ($S[4]$ and $S[9]$ are chosen).

3.2 Offline Query Phase

Before the offline query phase, we make sure the recorded flow information of all flows in the on-chip cache was dumped to the off-chip SRAM. For a proposed query on flow f, we need to extract relevant information from the SRAM counters and effectively recover the size of the considered flow. First, we relocate the corresponding mapped SRAM counters for f with k hash functions,

i.e., sub SRAM counters $S_f[r]$s ($r = 0, \ldots, k-1$), and extract k counter values. Since one or more counters among the k counters may also be reused by other flows, we need to apply a de-noising method to remove the impact of other flows and estimate the size of f. In this paper, we use two different estimation methods, namely Counter Sum estimation Method (CSM) and Maximum Likelihood estimation Method (MLM), to perform de-noising and estimation, which will be discussed in the following sections.

Furthermore, for the example shown in Figure 2, we have already known the final values of all the SRAM counters. If we estimate flow f_1, we use its flow ID f_1 to get its k mapped counters $S_{f1}[r]$s ($r = 0, \ldots, k-1$) (*i.e.*, $S[1]$, $S[3]$, and $S[4]$), and then calculate its estimation value according to the three counter values 1, 2 and 4.

4 THEORETICAL ANALYSIS ON CONSTRUCTION PHASE

In this section, we study the construction phase. We analyze the value of an arbitrary chosen counter $S_f[r]$ mapped to flow f. Generally, this counter value increases from two parts, flow f and other noising flows. We attempt to derive the expectation and variance of these two parts, respectively.

4.1 Some Fundamental Assumptions

We suppose the upper bound of flow size is N ($N \in \mathbb{Z}^+$). Based on past experience, we presume that the size of flows obeys a certain distribution, which means we have known the probability of an arbitrary flow with size of i is $\mathcal{P}_i$ ($i = 1, \ldots, N, \mathcal{P}_i \in [0, 1)$). Obviously, we have $\sum_{i=1}^{N} \mathcal{P}_i = 1$. Based on the known size distribution, we can calculate the expectation and variance of an arbitrary flow size z as follows,

$$\begin{cases} E(z) = \sum_{i=0}^{N} i \cdot \mathcal{P}_i = \mu \\ D(z) = \sum_{i=0}^{N} (\mu - i)^2 \cdot \mathcal{P}_i = \sigma^2. \end{cases} \tag{1}$$

In addition, the amount of flows in our test is large enough to approximately match the properties in Equation (1).

Suppose the maximum capacity of each cache entry is y, and the actual size of flow f is x, which satisfy

$$\begin{cases} x = p \cdot k + q, (q < k) \\ y = a \cdot k + b, (b < k). \end{cases} \tag{2}$$

Further, there are two situations that trigger cache eviction. One is $x \geq y$, which means the cache entry of flow f cannot continuously count if not evicted; and the other is $Q > M$, which means the cache table cannot accommodate all captured flows at the same time.

4.2 Analysis on Portion of Flow f Added to $S_f[r]$

We first focus on the addition from flow f to its arbitrary counter $S_f[r]$. Suppose flow f is evicted for t times in total, and its i-th evicted value can be modeled as a random variable E_i with value $e_i \in \{1, \ldots, y\}$. Clearly, it holds that

$$x = \sum_{i=1}^{t} e_i. \tag{3}$$

Therefore, $\{E_1, \ldots, E_t\}$ denotes a sequence of random integers (later we will prove their independence).

We map an evicted value of e_i to k SRAM counters, denoted as $S_f[r]$s ($r = 0, \ldots, k-1$). Suppose flow f's i-th eviction value E_i causes the counter $S_f[r]$ to increase EV_i, and we have $EV_i = EV_{i1} + EV_{i2}$, where EV_{i1} denotes the aliquot part and EV_{i2} denotes the remainder portion from E_i. Generally, let $e_i = ev_{i1} \cdot k + ev_{i2}$. Then, EV_{i1} is a variable with the certain value ev_{i1}, which means $E(EV_{i1}) = ev_{i1}$ and $D(EV_{i1}) = 0$. And EV_{i2} denotes a sum of ev_{i2} Bernoulli random variables, each of which takes the value 1 with probability $\frac{1}{k}$, making variable EV_{i2} follows a binomial distribution: $EV_{i2} \sim B(ev_{i2}, \frac{1}{k})$, which means

$$E(EV_{i2}) = \frac{ev_{i2}}{k}, D(EV_{i2}) = \frac{ev_{i2}}{k}(1 - \frac{1}{k}). \tag{4}$$

Therefore,

$$\begin{cases} E(EV_i) = ev_{i1} + \dfrac{ev_{i2}}{k} \\ D(EV_i) = \dfrac{ev_{i2}}{k}(1 - \dfrac{1}{k}). \end{cases} \tag{5}$$

First, we analyze parameter EV_{i2} and its value ev_{i2}. On the one hand, all packets from all flows can be regarded as arriving uniformly and with equal probability. On the other hand, whether LRU or random replacement algorithms select the cache entry to be evicted regardless of its stored value, then random variable sequence $\{E_1, \ldots, E_t\}$ can be regarded as independent and identically distributed. Consequently, for cases other than evicting y in a cache entry overflow, the probability p_i of evicting value i ($i = 1, \ldots, y-1$) should be equal approximately, that is,

$$p_1 \approx p_2 \approx \ldots \approx p_{y-1} = p. \tag{6}$$

Further, define p_y as the probability of evicting y, which indicates that its cache entry is overflowed, we have

$$\sum_{i=1}^{y} p_i = \sum_{i=1}^{y-1} p + p_y = 1. \tag{7}$$

In Section 6, the real network traces further verify that the flow sizes obey heavy tailed distribution as shown in [20, 41]; particularly, more than 92% flows are less than the average size of all captured flows. When the maximum capacity y is only the rough average flow size, even an overflow will hardly happen *i.e.*, $p_y \to 0$, and therefore, $p \approx \frac{1}{y-1}$. Then we have

$$\begin{aligned} E(EV_{i2}) &= \sum_{j=0}^{k-1} j \cdot P(EV_{i2} = j) \\ &= \sum_{j=0}^{b-1} j \sum_{i=0}^{a} p_{ik+j} + \sum_{j=b}^{k-1} j \sum_{i=0}^{a-1} p_{ik+j} + b \cdot p_y \\ &= \frac{ak^2 - ak + b^2 - b}{2} p + b \cdot p_y, \end{aligned}$$

where $b < k$ from Equation (2). Afterwards, we know empirical shared counter schemes perform well when parameter k is not too big (*e.g.*, 3), thus $k \ll y$, and then

$$E(EV_{i2}) \approx \frac{y(k-1)}{2} \cdot \frac{1}{y-1} + b \cdot 0 \approx \frac{k-1}{2}.$$

Next from Equation (4), we have

$$ev_{i2} = k \cdot E(EV_{i2}) \approx \frac{k(k-1)}{2}. \tag{8}$$

Second, we analyze the eviction time t. We have known that t evictions from flow f form an *i.i.d.* integer random variable sequence $E_1, \ldots, E_t$, whose sum is x, then we have

$$E(x) = E(E_i) \cdot E(t) \quad (i = 1, \ldots, t) \tag{9}$$

by [22]. Further, from Equations (6) and (7), we know

$$\begin{aligned} E(E_i) &= \sum_{j=1}^{y} j \cdot P(E_i = j) = \sum_{j=1}^{y} j \cdot p_j \\ &= \frac{y(y-1)}{2} p + y \cdot p_y = y[\frac{p(y-1)}{2} + p_y] \approx \frac{y}{2}, \end{aligned}$$

from Equation (9), then we have

$$E(t) = \frac{E(x)}{E(E_i)} = \frac{2x}{y}. \tag{10}$$

As we know that t evictions are independent of each other, we define a random variable Y as the addition of flow f to its arbitrary counter $S_f[r]$ $(r = 0, \ldots, k-1)$. Clearly, we have

$$Y = \sum_{i=1}^{t} EV_i. \tag{11}$$

Then, we obtain

$$E(Y) = \sum_{i=1}^{t} E(EV_i) = \sum_{i=1}^{t} (ev_{i1} + \frac{ev_{i2}}{k}) = \frac{1}{k} \sum_{i=1}^{t} e_i,$$

and from Equations (2) and (3), we have

$$E(Y) = \frac{x}{k} = p + \frac{q}{k}. \tag{12}$$

Similarly, we obtain

$$D(Y) = \sum_{i=1}^{t} D(EV_i) = \frac{1}{k}(1 - \frac{1}{k}) \sum_{i=1}^{t} ev_{i2}. \tag{13}$$

Finally, from Equations (8), (10), and (13), we have

$$D(Y) \approx \frac{(k-1)E(t)ev_{i2}}{k^2} \approx \frac{x(k-1)^2}{yk}. \tag{14}$$

4.3 Analysis on Portion of Flow $\overline{f}$ other than f Added to $S_f[r]$

In this subsection, we focus on the flows other than f that add to any chosen counter $S_f[r]$ of flow f. For any other flow, say $\overline{f}$, we denote z as its actual size and variable Z as its addition to $S_f[r]$, where $z = k \cdot z_1 + z_2$ $(z_1 = 0, \ldots, N-1, z_2 = 0, \ldots, k-1)$.

Suppose that we capture Q flows totally, meaning there are $Q-1$ $\overline{f}$s which map to their $Q-1$ logical $S_{\overline{f}}$s. Let Z_{total} be the aggregated addition of all $\overline{f}$s to $S_f[r]$. Obviously, flow $\overline{f}$ adds to $S_f[r]$ if and only if the counter $S_f[r]$ is exactly one of its k mapped SRAM counters, with probability $p_{select} = \frac{\binom{L-1}{k-1}}{\binom{L}{k}} \cdot \frac{1}{k} = \frac{1}{L}$. Suppose flow $\overline{f}$ is evicted for $t_{\overline{f}}$ times, indicating $Z = \sum_{i=1}^{t_{\overline{f}}} Z_i$. Similar to Equations (12) and (14), we know $E(Z) = p_{select} \cdot (z_1 + \frac{z_2}{k})$ and $D(Z) \approx p_{select} \cdot \frac{(z_1 \cdot k + z_2)(k-1)^2}{yk}$, then we have

$$\begin{aligned} E(Z_{total}) &= \sum_{z_1=0}^{N-1} \sum_{z_2=0}^{k-1} (\mathcal{P}_{z_1 \cdot k + z_2}) QE(Z) \\ &= \frac{Q}{Lk} \sum_{z_1=0}^{N-1} \sum_{z_2=0}^{k-1} (\mathcal{P}_{z_1 \cdot k + z_2})(z_1 \cdot k + z_2), \end{aligned}$$

further, from Equation (1) we have

$$E(Z_{total}) = \frac{Q\mu}{Lk}. \tag{15}$$

Similarly,

$$\begin{aligned} D(Z_{total}) &= \sum_{z_1=0}^{N-1} \sum_{z_2=0}^{k-1} (p_{z_1 \cdot k + z_2}) QD(Z) \\ &\approx \frac{Q(k-1)^2}{ykL} \sum_{z_1=0}^{N-1} \sum_{z_2=0}^{k-1} (z_1 \cdot k + z_2) \mathcal{P}_{z_1 \cdot k + z_2}, \end{aligned}$$

and from Equation (1) we have

$$D(Z_{total}) \approx \frac{Q\mu(k-1)^2}{ykL}. \tag{16}$$

4.4 Analysis on Value of $S_f[r]$

In conclusion, for a flow f of size $x = pk + q$ mapped to k off-chip SRAM counters $S_f[r]$s $(r = 0, \ldots, k-1)$, where any counter has the value of the final aggregated storage, say X, and then we have

$$X = Y + Z_{total}, \tag{17}$$

where Y is the addition of flow f to the counter $S_f[r]$ and Z_{total} is the total noises of other flows $\overline{f}$s to $S_f[r]$. From Equations (12), (14), (15), (16), and (17), we have

$$\begin{cases} E(X) = \frac{x}{k} + \frac{Q\mu}{Lk} \\ D(X) \approx \frac{x(k-1)^2}{yk} + \frac{Q\mu(k-1)^2}{ykL}. \end{cases} \tag{18}$$

5 THEORETICAL ANALYSIS ON QUERY PHASE

In this section, we introduce the query phase of CAESAR. We use two different methods to estimate the flow sizes, and analyze the estimated expectations and variances using two different methods. Furthermore, we derive their corresponding confidence intervals to characterize their accuracy.

5.1 Counter Sum Estimation (CSM)

We first use the Counter Sum Estimation (CSM) method, which is based on the moment estimation.

From Equation (18), we have

$$x = kE(X) - \frac{Q\mu}{L}. \tag{19}$$

As each evicted value of flow f is mapped to k SRAM counters, $S_f[r]$s $(r = 0, \ldots, k-1)$, we can use the values of these k counters

as samples and calculate $E(X)$ by CSM, *i.e.*, $E(X) = \frac{1}{k}\sum_{r=0}^{k-1} S_f[r]$. Then the estimated flow size of f is

$$\hat{x} = \sum_{r=0}^{k-1} S_f[r] - \frac{Q\mu}{L}. \tag{20}$$

Moreover, for all $r = 0, \ldots, k-1$, $S_f[r]$s are k independent sample values, and therefore, different samples result in different values of $\hat{x}$, then we have

$$\begin{aligned} E(\hat{x}) &= E(\sum_{j=0}^{k-1} S_f[j]) - \frac{Q\mu}{L} = \sum_{j=0}^{k-1} E(S_f[j]) - \frac{Q\mu}{L} \\ &= kE(X) - \frac{Q\mu}{L} = x. \end{aligned} \tag{21}$$

Thus, we claim that the estimation is unbiased.

It is clear that $D(\hat{x}) = k^2 D(X)$, and further combining with Equation (18), we have

$$D(\hat{x}) \approx \frac{xk(k-1)^2}{y} + \frac{Q\mu k(k-1)^2}{yL}. \tag{22}$$

As a binomial distribution can be approximated to a Gaussian distribution in the following way:

$$B(v, w) \to N(vw, vw(1-w)), \tag{23}$$

and then we have $EV_i \sim N(ev_{i1} + \frac{ev_{i2}}{k}, \frac{ev_{i2}}{k}(1-\frac{1}{k}))$ from Equation (5). Considering that all t random variables EV_is are independent, then from Equations (11), (12), and (14), we know $Y \sim N(\frac{x}{k}, \frac{x(k-1)^2}{yk})$. Note that here we use the conclusion that the sum of several normal distributed random variables still obeys a normal distribution. Similarly, Z_{total} can be approximately regarded as obeying a Gaussian distribution too, *i.e.*, $Z_{total} \sim N(\frac{Q\mu}{Lk}, \frac{Q\mu(k-1)^2}{ykL})$, and then

$$X \sim N(\mu_X, \Delta_X), \tag{24}$$

where $\mu_X = \frac{x}{k} + \frac{Q\mu}{Lk}, \Delta_X = \frac{x(k-1)^2}{yk} + \frac{Q\mu(k-1)^2}{ykL}$. From the unbiased estimation in Equation (21), we have

$$\hat{x} \sim N(x, \frac{xk(k-1)^2}{y} + \frac{Q\mu k(k-1)^2}{yL}). \tag{25}$$

Based on the properties of Gaussian distribution, the maximum confidence interval of CSM estimation is

$$\hat{x} \pm Z_\alpha \sqrt{\frac{xk(k-1)^2}{y} + \frac{Q\mu k(k-1)^2}{yL}}. \tag{26}$$

where α is the reliability.

5.2 Maximum Likelihood Estimation

Maximum Likelihood estimation Method (MLM) is another useful estimation method, which is much more complicated than CSM but more accurate. To reduce computational complexity while maintaining high accuracy, we use the fisher information $\mathcal{I}(\hat{x})$ [19] to approximate the flow estimation results as a Gaussian distribution.

From Equation (24), $X \sim N(\mu_X, \Delta_X)$, where $\mu_X = \frac{x}{k} + \frac{Q\mu}{Lk}$, $\Delta_X = \frac{x(k-1)^2}{yk} + \frac{Q\mu(k-1)^2}{ykL}$. Suppose evictions of flow f are hashed to k SRAM counters $S_f[r]$s ($r = 0, \ldots, k-1$), whose values can be denoted by random variables $W_1, \ldots, W_k$ with sample values $w_1, \ldots, w_k$, respectively. Then, the probability that the observed values of $(W_1, \ldots, W_k)$ are exactly $(w_1, \ldots, w_k)$ is approximately $V(p, q) = \prod_{i=1}^{k} \int f(w_i; p, q) dw_i$. Since $\prod_{i=1}^{k} dw_i$ is independent of the values of p and q, we only need to consider

$$V(p, q) = \prod_{r=1}^{k} f(w_i; p, q) = (\frac{1}{\sqrt{2\pi\Delta_X}})^k e^{-\frac{1}{2\Delta_X}\sum_{i=1}^{k}(w_i-\mu_X)^2}. \tag{27}$$

Next, we safely replace $V(p, q)$ with $\ln V(p, q)$ for the sake of analyzing the maximum value of $V(p, q)$. MLM aims to find the estimated size of flow f through maximizing the likelihood function $\ln V$, *i.e.*,

$$\hat{x} = \arg\max\{\ln V\}. \tag{28}$$

By taking the first derivative of $\ln V(p, q)$ and enforcing it to zero for maximum value computation, we obtain

$$x^2 + (\frac{2Q\mu}{L} + \frac{(k-1)^2}{y})x + (\frac{Q^2\mu^2}{L^2} + \frac{Q\mu(k-1)^2}{yL} - k\sum_{i=1}^{k} w_i^2) = 0,$$

and thereby calculate its solution as

$$\hat{x} = \frac{1}{2}(\sqrt{\frac{(k-1)^4}{y^2} + 4\sum_{i=1}^{k} kw_i^2} - \frac{2Q\mu}{L} - \frac{(k-1)^2}{y}).$$

According to the theory of MLM [27], $\hat{x}$ can be approximated to a Gaussian distribution when L is sufficiently large, *i.e.*,

$$\hat{x} \sim N(x, \frac{1}{\mathcal{I}(\hat{x})}), \tag{29}$$

$$\mathcal{I}(\hat{x}) = -E(\frac{d^2 \ln(V)}{dx^2}). \tag{30}$$

From Equation (27), we know $\frac{d \ln V}{dx} = \frac{\Delta'_X}{2\Delta_X^2}(w_i - \mu_X)^2 + \frac{\mu'_X}{\Delta_X}(w_i - \mu_X) - \frac{\Delta'_X}{2\Delta_X}$ and $\frac{d^2 \ln V}{dx^2} = -\frac{\Delta'^2_X}{\Delta_X^3}(w_i-\mu_X)^2 - (\Delta'_X+1)(\frac{\mu'_X}{\Delta_X^2})(w_i-\mu_X) - \frac{\mu'^2_X}{\Delta_X} + \frac{\Delta'^2_X}{2\Delta_X^2}$, where $\mu'_X = \frac{1}{k}$ and $\Delta'_X = \frac{(k-1)^2}{yk}$. As $E(\mu_X - w_i) = 0$, then $E[(\mu_X - w_i)^2] = \Delta_X$, and $E(\frac{d^2 \ln V}{dx^2}) = -\frac{\Delta'^2_X + 2\mu'^2_X \Delta_X}{2\Delta_X^2}$. From Equations (29) and (30),

$$D(\hat{x}) = \frac{1}{\mathcal{I}(\hat{x})} = \frac{2k^2\Delta_X^2}{2\Delta_X + \frac{(k-1)^4}{y^2}}, \Delta_X = \frac{x(k-1)^2}{yk} + \frac{Q\mu(k-1)^2}{ykL}.$$

Finally, we have

$$\hat{x} \sim N(x, \frac{2k^2\Delta_X^2}{2\Delta_X + \frac{(k-1)^4}{y^2}}) \tag{31}$$

and the maximal confidence interval of MLM estimation is

$$\hat{x} \pm Z_\alpha \sqrt{\frac{2k^2\Delta_X^2}{2\Delta_X + \frac{(k-1)^4}{y^2}}}, \tag{32}$$

where $\Delta_X = \frac{x(k-1)^2}{yk} + \frac{Q\mu(k-1)^2}{ykL}$ and α is the reliability.

6 EVALUATION

In this section, we conduct both software simulations using C++ and FPGA experiments to validate the performance of the proposed CAESAR scheme. We compare it to two state-of-the-art schemes, *i.e.*, Cahce-Assisted Stretchable Estimator (CASE) [24] and Randomized Counter Sharing (RCS) [20].

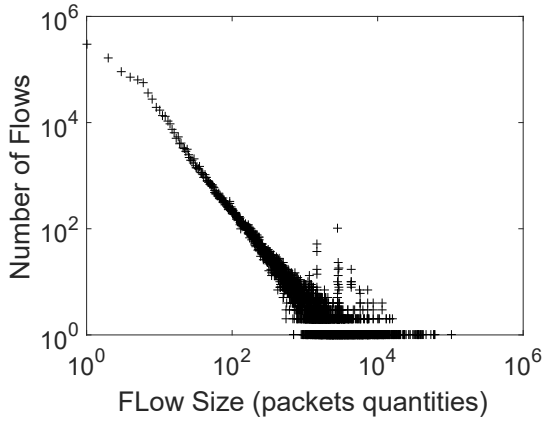


Figure 3: Heavy tailed distribution of flow size

6.1 Real Network Traffic Distribution

We capture real-world network traces, including TCP, UDP, and ICMP flows on a 10Gbps link of a backbone router. After capturing each packet, we extract the information of the 5-tuple packet header to artificially generate its unique flow ID, using SHA-1 and APHash functions. We captured about $n = 27,720,011$ packets, which totally generate $1,014,601$ different flows IDs. Easily, we calculate the actual sizes of all these flows and plot size distribution in Figure 3. We can see the flows follow a heavy tailed distribution.

6.2 Experimental Setup

For software simulations, we first set up a cache table containing M cache entries of capacity of y, then the cache memory is $\frac{M \cdot log_2 y}{1024 \times 8}$ KB. The coarse average flow size is known and is given by $\frac{n}{Q}$. We set $y = \lfloor 2 \cdot \frac{n}{Q} \rfloor$, and based on the heavy tailed distribution, the proportion of network flows smaller than y exceeds 95%, which means the cache entries hardly overflow. In addition, we set the cache size to only 97.66 KB (millions of flows performs well, may need scale to larger with a larger scale routing infrastructure) in CASE and CAESAR while RCS has no cache memory cost. Similarly, we suppose the off-chip SRAM table contains L counters with uniform capacity of l, and the SRAM size is then $\frac{L \cdot log_2 l}{1024 \times 8}$ KB.

For FPGA experiments, we model CAESAR, CASE, and RCS in VHDL code verified by simulator ModelSim SE-64 10. 1c, using Xilinx Foundation ISE 14.3 [10, 28]. Then we implement them on a Xilinx Virtex-7 FPGA chip platform [31]. Its block RAM size is 68 MB and its peak serial bandwidth at full duplex is 2, 784 Gbps. Moreover, its Dual Port RAM supports duplex reading and writing in the fast cache to improve work efficiency. Our maximal design clock rate is 18.912 MHz, input of packets' IDs is 36-bit bus with the same clock. Therefore, it supports streams up to 680.832 Mbps.

6.3 Simulation Results

6.3.1 CAESAR Evaluation. We not only consider both CSM and MLM estimation methods, but also performance both LRU and random replacement algorithms for CAESAR.

We compare the two estimation methods of CAESAR in Figure 4. Figure 4(a) for CSM and Figure 4(b) for MLM are the estimated-actual plots under the same SRAM size of 91.55 KB. A point in the plot represents a flow, where its y coordinate denotes the estimated flow size and x coordinate is the known actual flow size. Moreover, we use a reference line, *i.e.*, $y = x$, to reflect the accuracy of estimation results. We can see that the CAESAR can estimate flow sizes accurately even when the SRAM size is less than 100 KB, and CSM and MLM estimation results have little difference. Furthermore, the average relative errors for certain flow sizes in Figure 4(d) are slightly closer to 0 than errors in Figure 4(c), which indicates that MLM estimation is slightly more accurate than CSM estimation in CAESAR scheme, especially for smaller flows. Then, we choose CSM as the default estimation method for CAESAR.

6.3.2 Compared with CASE. Figure 5 shows the CASE estimation results. We first set the SRAM size to 183.11 KB. Figures 5(a) and 5(c) show that the estimated flow sizes of CASE are almost 0, resulting in relative errors close to 100%, as shown in Figure 5(c). Because under the same SRAM size $\frac{L \cdot log_2 l}{1024 \times 8}$ KB, L of CASE must be at least equal to Q, while L of CAESAR can be much less than Q, making l of CAESAR much larger than l of CASE to store large flows better. Then, we increase the SRAM size to 1.21 MB, expanding l about six times. Figures 5(b) and 5(d) show CASE is slightly improved, a small portion of flows can be estimated accurately while the others are still bad.

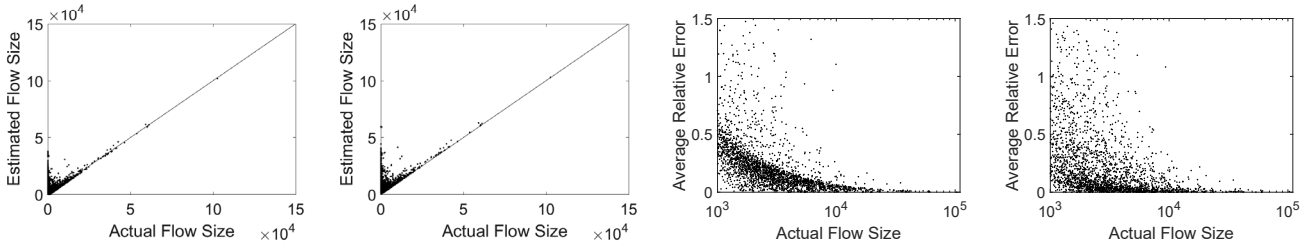
6.3.3 Compared with RCS. Note that RCS is cache-free.

First, Figure 6 shows the RCS estimation results under the same SRAM size of Figure 4, with the assumption that the off-chip SRAM is so fast that no packets are failed to be captured (*i.e.*, lossless assumption but not real). The results in Figures 4(a) and 4(b) are quite similar with that in Figures 6(a) and 6(b), respectively, as well as Figure 6(d) with Figure 4(d), all of which imply CAESAR still works accurately from another perspective, *i.e.*, the cache size is very small as $y = 1$. Figure 6 does not contain MLM results of RCS as in Figure 4(d) because its binary search is extremely slow.

Second, Figure 7 presents the actual results for the RCS method without the lossless assumption. The loss rate is set to $\frac{2}{3}$ for Figures 7(a) and 7(c), and $\frac{9}{10}$ for Figures 7(b) and 7(d), which is based on the empirical speed difference between the on-chip cache and off-chip SRAM [24]. Furthermore, the corresponding average relative errors of RCS as shown in Figures 7(c) (on average 67.68%) and 7(d) (on average 90.06%) mostly exceed 50%, which are much worse than that of CAESAR in Figures 4(c) (on average 25.23%) and 4(d) (on average 30.83%).

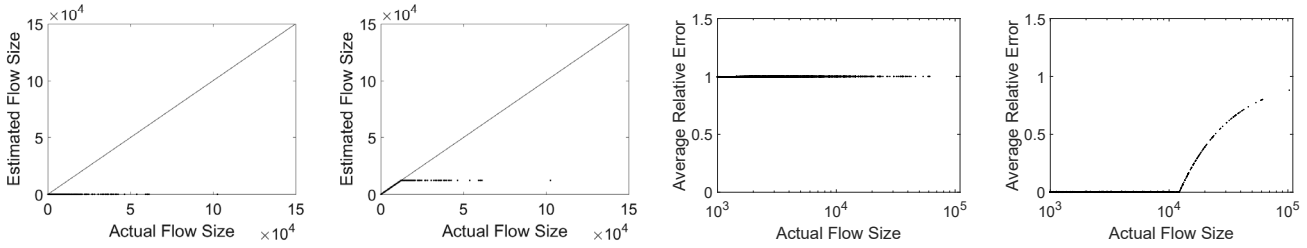
6.4 Experimental Results

Under the same SRAM size constraint of 183.11 KB, we compare the time efficiency of RCS, CASE, and CAESAR as shown in Figure 8. When processing packets smaller than 10000, CASE is more time-consuming than RCS and CAESAR due to its high computational cost of power operations. When processing packets larger than 10000 , the processing time of RCS drastically increases and exceeds CASE, resulting in packet loss caused by slow update of off-chip counters. Obviously, CAESAR is always more time efficient than the other two schemes. In particular, it is on average 74.8% and up



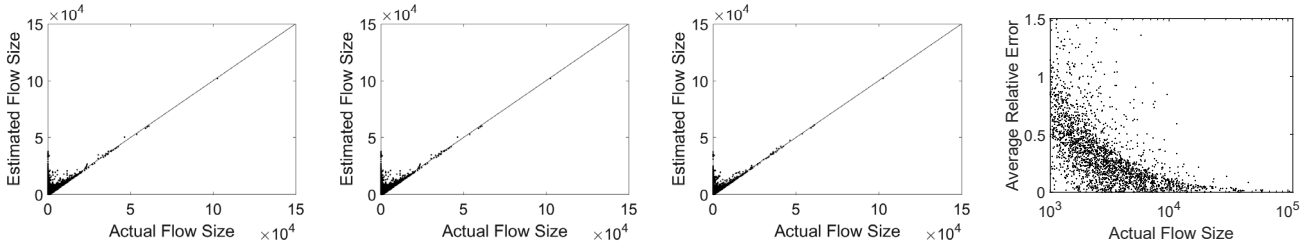
(a) CSM, SRAM=91.55KB (b) MLM, SRAM=91.55KB (c) CSM, SRAM=183.11KB (d) MLM, SRAM=183.11KB

Figure 4: Estimated flow size vs. actual flow size ((a) and (b)); average relative error vs. actual flow size ((c) and (d)) for CAESAR



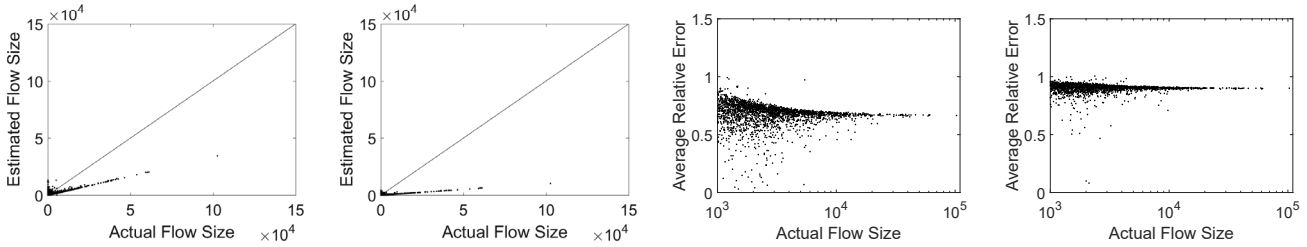
(a) SRAM=183.11KB (b) SRAM=1.21MB (c) SRAM=183.11KB (d) SRAM=1.21MB

Figure 5: Estimated flow size vs. actual flow size ((a) and (b)); average relative error vs. actual flow size ((c) and (d)) for CASE



(a) CSM, SRAM=91.55KB (b) MLM, SRAM=91.55KB (c) CSM, SRAM=183.11KB (d) CSM, SRAM=183.11KB

Figure 6: Estimated flow size vs. actual flow size ((a), (b), and (c)); average relative error vs. actual flow size ((d)) for RCS under lossless assumption



(a) loss 2:3, SRAM=183.11KB (b) loss 9:10, SRAM=183.11KB (c) loss 2:3, SRAM=183.11KB (d) loss 9:10, SRAM=183.11KB

Figure 7: Estimated flow size vs. actual flow size ((a) and (b)); average relative error vs. actual flow size ((c) and (d)) for RCS under realistic loss assumption

to 92.4% faster than CASE, while on average 75.5% and up to 90% faster than RCS. Moreover, this FPGA chip is still too slow, which need to be improved in the future.

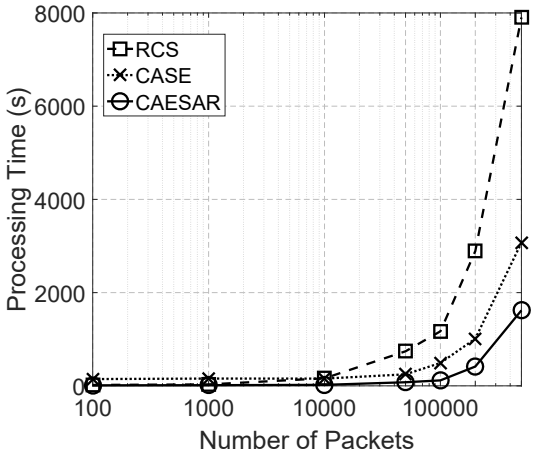


Figure 8: Processing time vs. number of packets

7 CONCLUSION

The main contribution of this paper is to propose Cache Assisted and randomizEd ShAring counteRs (CAESAR) for per-flow size estimation. The key improvements of CAESAR over the prior arts are high space efficiency, high time efficiency, and high estimation accuracy. This paper theoretically proves the estimation accuracy of CAESAR for both CSM and MLM methods, as well as implements simulations and FPGA experiments using real-world network traces to verify the advantages of CAESAR over CASE and RCS.

ACKNOWLEDGMENT

This work is partially supported by the NSFC Grants (No. 61502229, No. 61373130, No. 61472184, No. 61321491), the Fundamental Research Funds for the Central Universities (021014380079), and the Jiangsu Innovation and Entrepreneurship (Shuangchuang) Program.

REFERENCES

[1] R. Basat, G. Einziger, R. Friedman, and Y. Kassner. 2017. Optimal elephant flow detection. *arXiv preprint arXiv:1701.04021* (2017).
[2] M. Chen, S. Chen, and Z. Cai. 2017. Counter tree: A scalable counter architecture for per-flow traffic measurement. *IEEE/ACM TON* 25, 2 (2017), 1249–1262.
[3] S. Chen, M. Chen, and Q. Xiao. 2017. Per-Flow Size Measurement. In *Traffic Measurement for Big Network Data*. Springer, 11–45.
[4] H. Dai, M. Li, and A. Liu. 2018. Finding Persistent Items in Distributed Datasets. In *IEEE INFOCOM*.
[5] H. Dai, M. Shahzad, A. Liu, and Y. Zhong. 2016. Finding persistent items in data streams. *VLDB* 10, 4 (2016), 289–300.
[6] E. Demaine, A. López-Ortiz, and J. Munro. 2002. Frequency estimation of internet packet streams with limited space. *Algorithms¡ªESA* (2002), 11–20.
[7] N. Duffield, C. Lund, and M. Thorup. 2005. Estimating flow distributions from sampled flow statistics. *IEEE/ACM TON* 13, 5 (2005), 933–946.
[8] G. Einziger, B. Fellman, and Y. Kassner. 2015. Independent counter estimation buckets. In *IEEE INFOCOM*. 2560–2568.
[9] C. Estan and G. Varghese. 2003. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM TOCS* 21, 3 (2003), 270–313.
[10] Mentor Graphics. 2017. Modelsim user manual. (2017). https://www.mentor.com/products/fv/modelsim.html.
[11] F. Hao, M. Kodialam, and T. V. Lakshman. 2004. ACCEL-RATE:a faster mechanism for memory efficient per-flow traffic estimation. *ACM SIGMETRICS* 32, 1 (2004), 155–166.
[12] C. Hu, B. Liu, H. Zhao, K. Chen, Y. Chen, C. Wu, and Y. Cheng. 2010. Disco: Memory efficient and accurate flow statistics for network measurement. In *IEEE ICDCS*. 665–674.
[13] C. Hu, S. Wang, J. Tian, B. Liu, Y. Cheng, and Y. Chen. 2008. Accurate and efficient traffic monitoring using adaptive non-linear sampling method. In *IEEE INFOCOM*. 26–30.
[14] A. Kabbani, M. Alizadeh, M. Yasuda, R. Pan, and B. Prabhakar. 2010. Approximate fairness with quantized congestion notification for multi-tenanted data centers. In *IEEE HOTI*. 58–65.
[15] N. Kamiyama and T. Mori. 2006. Simple and Accurate Identification of High-Rate Flows by Packet Sampling. In *IEEE INFOCOM*. 1–13.
[16] R. M. Karp, S. Shenker, and C. H. Papadimitriou. 2003. A simple algorithm for finding frequent elements in streams and bags. *ACM TODS* 28, 1 (2003), 51–55.
[17] M. Kodialam, T. V. Lakshman, and S. Mohanty. 2004. Runs based traffic estimator (RATE): A simple, memory efficient scheme for per-flow rate estimation. *IEEE INFOCOM* 3 (2004), 1808–1818.
[18] A. Kumar, M. Sung, J. J. Xu, and J. Wang. 2004. Data streaming algorithms for efficient and accurate estimation of flow size distribution. In *ACM SIGMETRICS*. 177–188.
[19] E. Lehmann and G. Casella. 2006. *Theory of point estimation*. Springer Science & Business Media.
[20] T. Li, S. Chen, and Y. Ling. 2011. Fast and compact per-flow traffic measurement through randomized counter sharing. In *IEEE INFOCOM*. 1799–1807.
[21] T. Li, S. Chen, and Y. Ling. 2012. Per-flow traffic measurement through randomized counter sharing. *IEEE/ACM TON* 20, 5 (2012), 1622–1634.
[22] X. Li. 1996. *Foundations of Probability Theory (second edition)*. Higher Education Press. 214–249 pages.
[23] X. Li, F. Bian, M. Crovella, C. Diot, R. Govindan, G. Iannaccone, and A. Lakhina. 2006. Detection and identification of network anomalies using sketch subspaces. In *ACM SIGCOMM*. 147–152.
[24] Y. Li, H. Wu, T. Pan, H. Dai, J. Lu, and B. Liu. 2016. Case: Cache-assisted stretchable estimator for high speed per-flow measurement. In *IEEE INFOCOM*. 1–9.
[25] Y. Lu, A. Montanari, and B. Prabhakar. 2008. Counter braids: Asymptotic optimality of the message passing decoding algorithm. In *IEEE ALLERTON*. 209–216.
[26] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani. 2008. Counter braids: a novel counter architecture for per-flow measurement. *ACM SIGMETRICS* 36, 1 (2008), 121–132.
[27] J. Myung and D. Navarro. 2004. *Information matrix*. John Wiley & Sons, Ltd.
[28] Mehta N. 2011. Xilinx 7 Series FPGAs: User Guide Lite. (2011). http://www.eetimes.com/document.asp?doc_id=1278724.html.
[29] R. Stanojevic. 2007. Small active counters. In *IEEE INFOCOM*. 2153–2161.
[30] E. Tsidon, I. Hanniel, and I. Keslassy. 2012. Estimators also need shared values to grow together. In *IEEE INFOCOM*. 1889–1897.
[31] XILINX. 2017. 7 Series FPGAs Data Sheet: Overview. (2017). http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf.
[32] T. Yang, S. Gao, Z. Sun, Y. Wang, Y. Shen, and X. Li. 2018. Diamond Sketch: Accurate Per-flow Measurement for Real IP Streams. In *IEEE INFOCOM (Poster)*.
[33] T. Yang, J. Gong, H. Zhang, L. Zou, L. Shi, and X. Li. 2018. HeavyGuardian: Separate and Guard Hot Items in Data Streams. In *ACM SIGKDD*.
[34] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig. 2018. Elastic Sketch: Adaptive and Fast Network-wide Measurements. In *ACM SIGCOMM*.
[35] T. Yang, A. Liu, M. Shahzad, D. Yang, Q. Fu, G. Xie, and X. Li. 2017. A Shifting Framework for Set Queries. *IEEE/ACM TON* PP, 99 (2017), 1–16.
[36] T. Yang, L. Liu, Y. Yan, M. Shahzad, Y. Shen, X. Li, B. Cui, and G. Xie. 2017. SF-sketch: A Fast, Accurate, and Memory Efficient Data Structure to Store Frequencies of Data Items. In *IEEE ICDE*.
[37] T. Yang, L. Wang, Y. Shen, M. Shahzad, Q. Huang, X. Jiang, K. Tan, and X. Li. 2018. Empowering Sketches with Machine Learning for Network Measurements. In *ACM SIGCOMM workshop on NetAI*.
[38] T. Yang, H. Zhang, H. Wang, M. Shahzad, Q. Xin, X. Liu, and X. Li. 2018. FID-sketch: An Accurate Sketch to Store Frequencies in Data Streams. *World Wide Web Journal* (2018).
[39] T. Yang, Y. Zhou, H. Jin, S. Chen, and X. Li. 2017. Pyramid Sketch: a Sketch Framework for Frequency Estimation of Data Streams. *VLDB* 10, 11 (2017).
[40] Y. Zhang, L. Breslau, V. Paxson, and S. Shenker. 2002. On the characteristics and origins of internet flow rates. *ACM SIGCOMM* 32, 4 (2002), 309–322.
[41] Y. Zhou, Y. Zhou, M. Chen, Q. Xiao, and S. Chen. 2017. Highly compact virtual counters for per-flow traffic measurement through register sharing. In *IEEE GLOBECOM*. 1–6.
[42] Y. Zhou, Y. Zhou, S. Chen, and Y. Zhang. 2017. Per-flow counting for big network data stream over sliding windows. In *IEEE/ACM IWQOS*. 1–10.