Recyclable Counter With Confinement for Real-Time Per-Flow Measurement

DaeHun Nyang, *Member, IEEE*, and DongOh Shin, *Member, IEEE*

***Abstract*—With the amount of Internet traffic increasing substantially, measuring per-flow traffic accurately is an important task. Because of the nature of high-speed routers, a measurement algorithm should be fast enough to process every packet going through them, and should be executable with only a limited amount of memory, as well. In this paper, we use two techniques to solve memory/speed constraints: (1) recycling a memory block by resetting it (for memory constraint), and (2) confinement of virtual vectors to one word (for speed constraint). These techniques allow our measurement algorithm, called a recyclable counter with confinement (RCC), to accurately measure all individual flow sizes with a small amount of memory. In terms of encoding speed, it uses about one memory access and one hash computation. Unlike other previously proposed schemes, RCC decodes very quickly, demanding about three memory accesses and two hash calculations. This fast decoding enables real-time detection of a high uploader/downloader. Finally, RCC's data structure includes flow labels for large flows, so it is possible to quickly retrieve a list of large-flow names and sizes.**

***Index Terms*—Computer networks, network traffic measurement, computer network security, system analysis and design.**

I. Introduction

INTERNET traffic has substantially increased, and Cisco Systems Inc. projects the amount of IP traffic in 2017 will be 120,643 petabytes per month [1]. While this fast growth in volume necessarily accelerates development of high-speed routers, there has been a consistent need to measure the size of the traffic, not just roughly estimate the total traffic. Every individual flow size must be measured quickly and accurately, not only for accounting but also for security. Heavy downloaders/uploaders may be attackers that want to interfere with a network or overload a server by mounting a denial-of-service (DoS) attack, or they may be spammers. To be able to detect this at an early stage or in real time will greatly enhance not only availability but also the security of the network and the server. However, because of highly constrained requirements in terms of speed and memory, it is quite challenging to accurately measure individual flow sizes on a high-speed router with little memory, even if it is fast.

The simple approach (to account for each source by storing flow ID, counter pairs in a table) is not feasible considering that the number of flows may be too big to be counted in the table, e.g., 264 K sources for a one-minute Cooperative Association for Internet Data Analysis (CAIDA) dataset [4]. Because of the speed of routers, the data structure that holds counting values for flows is stored in an expensive, but fast memory, like SRAM, so the available amount of memory is not large. Therefore, the goal of a per-flow measurement algorithm is to measure individual flow sizes as accurately as possible with a small amount of memory. Also, updating a counter should be fast enough to be run on a high-speed router. For real-time detection of heavy flows, online decoding is also required. To be able to decode online, the number of memory accesses and hash computations for the estimation should be small enough to be performed on a high-speed router.

A lot of research on per-flow measurement has been conducted using various primitives, such as Bloom filters, coding techniques, Flajolet-Martin (FM) sketches, and randomized counter sharing [16], [20]–[22], [24]. To decrease memory usage, most researchers have used statistically shared counters, matrices, and Bloom filters. At the time of estimation, statistical noise is removed from each estimation. To make the estimation better, sometimes maximum likelihood estimation is adopted, but it involves a large amount of computation.

In this paper, we propose a per-flow measurement algorithm called a recyclable counter with confinement (RCC) that has both deterministic counting and probabilistic counting segments. RCC's deterministic counting segment is a simple accumulator, and its probabilistic segment greatly depends on linear counting and compact spread estimator (CSE) [12], [23]. Linear counting and CSE were originally for counting distinct elements in a set and multiple sets, respectively, while per-flow measurement algorithms were for counting multiplicity in a multiset. Instead of a deterministic hash on a destination IP, a randomized hash is used to adapt them to our algorithm. Randomized linear counting recycles its memory via repeated resetting when it becomes full from a large flow. RCC has many advantages over existing counting schemes. First, its estimation accuracy is higher than those of schemes recently proposed, such as the counter sharing method(CSM) [24] and the probabilistic multiplicity counting (PMC) [22], over all ranges of flow sizes, given the same amount of memory. For example, using only 50% of the memory of PMC, RCC gives better estimation accuracy. Second, RCC's encoding speed is comparable to CSM/maximum likelihood estimation

Manuscript received August 05, 2014; revised December 23, 2014; accepted December 06, 2015; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor A. Bremler-Barr. Date of publication January 18, 2016; date of current version October 13, 2016. This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant No. 2014R1A1A2059852 and Inha University. *(Corresponding author: DaeHun Nyang.)*

The authors are with the School of Computer and Information Engineering of Inha University, Incheon, Korea (e-mail: nyang@inha.ac.kr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2016.2514523

method (MLM)'s and PMC, requiring only 1.07 to 1.21 hash computations and memory accesses. Third, online decoding is possible with RCC, because RCC decoding for a flow requires only about two memory accesses and two hash computations, whereas other schemes require hundreds or thousands of memory accesses and hash computations, or very time-consuming maximum likelihood estimation. Fourth, RCC has a built-in data structure for flow-labeling of large flows, which is counted in total memory usage. Thus, it directly retrieves the list of large-flow labels and their sizes. We believe that built-in flow labeling working together with online decoding will lead per-flow measurement to new real-time services, such as real-time detection of heavy downloaders/uploaders, spammers and DoS attackers.

Our paper is organized as follows: After the introduction, we review related works in Section II, especially CSE in detail. Section III sketches our ideas. In Section IV, we describe our proposal, and Section V analyzes its accuracy. In Section VI, experiment results are shown, including a comparative study with CSM, MLM and PMC. Finally, we conclude in Section VII.

II. Related Works

Though packet sampling is an easy way to measure a large amount of traffic, measurement accuracy of Cisco's Netflow [2] and sFlow [3], which fall into the packet sampling category, are not good for an individual flow, especially for small flows. They have very large variance for small flows [22]. Recent research avoids packet sampling because of its large error in measurement, but examines all packets. RCC also examines all packets instead of packet sampling during a measurement period.

In order to identify elephant flows which can be defined as those flows that send more than a certain threshold (say 0.1% of the link capacity) during a measurement period, Estan and Varghese [13] presented a method in 2002 that was followed by several other researchers [15], [17], [18]. The basic idea in their schemes is to use a filter consisting of multiple, but duplicate, counters to accurately identify elephant flows. Their interest is not in measuring individual flows, but in identifying the elephants. Estan and Varghese's proposal is similar to RCC in that it uses two different structures to identify elephants, but RCC's goal is to measure all the flows.

Spectral Bloom filters (SBF) [14] and the multi-resolution space-code Bloom filter (MRSCBF) [16] use a Bloom filter to estimate individual flow sizes. SBF and MRSCBF are rather slow in terms of the number of memory accesses and hash computations, and also their estimation accuracy with a small amount of memory needs to be improved. For decoding, MRSCBF is very slow because it requires maximum likelihood estimation or mean value estimation. RCC is very fast, and has greater accuracy. To enhance Bloom filters' false positive rates, several approaches such as [25], [26] were proposed, but not for counting; rather, they are for dynamic set representation.

Counter braid (CB) encodes and stores a count value in the context of coding theory in order to compress the value [20]. It has multiple layers for counting from small numbers to large numbers. Owing to its high compression capability, it achieves high accuracy, although it fails to count flows with very limited memory [24]. Also, CB's encoding speed is still slow, and decoding should be executed offline in a batch way, that is, not just individual flow sizes, but all the flow sizes, are calculated in batch processing in a high-performance computer. RCC is fast in both encoding and decoding, and it is also able to decode an individual flow.

For per-flow measurement, probabilistic multiplicity counting (PMC), proposed in 2010, is the most comparable scheme to RCC [22]. PMC has a hybrid structure, so it uses an FM sketch [11] to measure large flows, and HitCounter [12] for small flows. Though it has much better measurement accuracy in measurement than MRSCBF and sFlow, RCC outperforms PMC, which uses twice the amount of memory of RCC, in terms of accuracy over all ranges of flow size over all memory sizes. This will be shown in Section VI.

In 2011, Li *et al.* proposed a randomized counter sharing method to measure individual flows accurately [24]. For encoding, it splits a flow size into 50 to 1,000 roughly equal shares, each of which is stored in one randomly chosen counter. Flow measurement or decoding is the summation of multiple but independent counters, subtracting the noise introduced by other flows that happen to share the counters. Its encoding is very fast, and it works well, even with an extremely small amount of memory. It showed that for traffic data with 10 million packets, the estimator successfully measured flow sizes with only 2 Mb ($=$ 512 KB) of memory, while CB and MRSCBF failed. CSM, however, is not good at measuring flows larger than a set threshold value because of its static data structure. More specifically, CSM has a fixed and identical number of shared counters for each flow, and thus, it has an upper limit of measurement. By increasing the number of counters, it can increase the upper limit, but at the expense of measurement accuracy. MLM, a maximum likelihood version of CSM, was proposed to improve measurement accuracy, but the decoding requires a substantially longer time than CSM.

RCC is different from previous methods, such as SBF, MRSCBF, CSM and PMC, in that a virtual memory block allocated to a flow in RCC is recyclable and is confined to one memory block. These two differences enhance the estimation accuracy and the speed.

Though compact spread estimator is not a multiplicity counter but a cardinality counter (RCC is a multiplicity counter), we review CSE in more detail because RCC is based upon CSE for its probabilistic counting. CSE was proposed by Yoon *et al.* in 2011 [23], and counts the number of distinct destinations for each source, while all the above-mentioned works count the multiplicity in a multiset. CSE counts cardinalities (spread or fan-out, equivalently) of multiple sets, and it is based on linear counting [12]. Thus, CSE can be viewed as a multiple-set version of linear counting. Every individual source has its own virtual vector for linear counting, some portion of which is randomly shared with another source's virtual vector. CSE uses an m-bit array, B, for estimation. Whenever a packet arrives, its corresponding virtual vector is chosen according to its source address, and the encoder sets a bit according to its destination address on the virtual vector: $B[H(src|(H(dst) \bmod s))] \leftarrow 1$, where s is the size of the virtual vector. $\hat{k}$, the estimation of a source's cardinality, is the estimation of linear counting for the virtual vector subtracted by the average noise

over the entire memory space, which is also calculated by linear counting as follows:

$$\hat{k} = s \cdot \ln(V_m) - s \cdot \ln(V_s), \quad (1)$$

where V_m and V_s are the fractions of 0's in B and in the virtual vector, respectively. The first term (negative) is the average noise over B, and the second term (positive) is the estimation of the virtual vector. Both estimations are from linear counting.

III. Ideas

In this section, we present two ideas to enhance accuracy and speed of the per-flow estimation: a small-but-recyclable counter and confinement of the virtual vector.

A. Small-but-Recyclable Counter for Accuracy

There are two important factors that determine the accuracy of traffic measurement schemes: saturation speed and the memory block size for a flow.

It is obvious that as time goes on during a measurement period, the number of packets increases, and the memory becomes saturated with too many 1's (or full counters) in most counting algorithms. The more 1's there are, the less accurate the estimation. We note that a Bloom filter works well until 50% of the memory is full (the golden ratio), and linear counting works well until 70% of the memory is full [12]. Thus, the filling speed of 1's is an important factors that decides a measurement algorithm's accuracy when there is limited memory. If the filling speed is fast with a given memory size, the memory will soon be saturated and the accuracy degrades quickly. Therefore, it is better for accuracy if there is a way to delay saturation.

To reduce the amount of memory required for measurement of large numbers of flows, it is also necessary for each flow to share memory with other flows, so most measurement schemes have adopted random sharing of memory among flows [16], [20]–[22], [24]. In measurement algorithms, a memory block is allocated to each flow for measurement, which is called a virtual matrix in PMC and MRSCBF, and a virtual vector in CSM and CSE. To reduce memory, some portion of a memory block is shared among multiple flows. Most measurement algorithms allocate the fixed and same size of memory block to each and every flow, because when the first packet of a flow arrives, an algorithm cannot decide whether it will become a large flow or a small flow. When a measurement scheme estimates traffic with large flows, the block size should be large enough to accommodate large flows. Because of the random memory sharing among flows, a larger block size implies a greater shared portion among flows. To share more memory gives better memory utilization, but it also causes less accuracy in estimation because the amount of noise introduced into a block is in proportion to the block size. In particular, small flows that would fit in a small block suffer from more noise than their actual size warrants. However, the block size cannot be set too small, to minimize noise, considering that, these days, the volume of traffic from a flow can grow sharply owing to various multimedia data. Let us take an example from PMC. It uses a very large virtual probabilistic counting with stochastic averaging (PCSA) matrix, where the size ranges from 32×32 to 32×256. Because of this large memory block, an FM sketch (the main component of PMC) is not good at counting small flows, and therefore, PMC adopts linear counting only for small flows by collapsing the matrix into a bit vector, which has the effect of making the memory block smaller [22]. Another example is CSM, where the number of 6–9 bit counters, l ($= 50 - 1,000$), determines the size of a memory block. When l is large, it can count large flows with less accuracy, but when it is small, it can count only up to a small size, but with better accuracy, because the small memory block is quickly saturated. Consequently, although we can control the block size, the control allows only a trade-off between accuracy and the upper limit of the counting range.

Our approach for better estimation is (1) *to recycle memory blocks by repeated resetting to prevent saturation*, and (2) *to use a very small memory block for each flow for better accuracy*. Obviously, recycling memory blocks distorts all the information that a memory block holds, and a small memory block has a small estimation upper limit, although those may provide slow saturation and better accuracy. To resolve these problems, we use a hash table to accumulate flow sizes measured by a memory block. Whenever a memory block for a flow reaches its maximum capacity, its estimation result is accumulated in the hash table (therefore, the information contributed by the flow is preserved in the table), and then the memory block is recycled after resetting the block to the average noise level (therefore, the information contributed by the other flows is preserved in the small memory block). Because the majority of flows are small enough to fit in a memory block, the number of flows to be managed by the hash table is not large, compared to the total number of flows. Recycling with resetting sharply contrasts with previously proposed schemes in that the number of 1's in RCC fluctuates, whereas the number in other schemes is ever-increasing. Repeated resetting prevents saturation, and thus, estimation accuracy is enhanced.

For the probabilistic counting structure with small memory blocks, we use a modified CSE so that it uses a randomized hash function instead of hashing deterministically over a destination address. By this randomization, different bit positions can be set to 1, even for multiple packets from the same flow. Because CSE was originally devised to count the spread of a source, it is good at counting small numbers, but not good at counting large numbers. In our approach, an approximation counting scheme that estimates small numbers well is desirable because we have small memory blocks and an independent accumulator. Whenever 70% of a virtual vector is full of 1's, its estimation value is accumulated in a hash table, and the vector is reset. Here, 70% is the percentage at which the linear counting works well according to Whang *et al.* [12]. However, because of the large number of accumulations for a large flow, a small error in each accumulation might be amplified. Because of this, instead of directly using their approximation formula shown in (1), we use a better estimation formula derived from Whang *et al.* [12].

B. Confinement of Virtual Vector for Speed

Another aspect of the performance of a measurement algorithm is encoding/decoding speed. For faster encoding and decoding with the randomized counter, we devised a technique called confinement of the virtual vector.

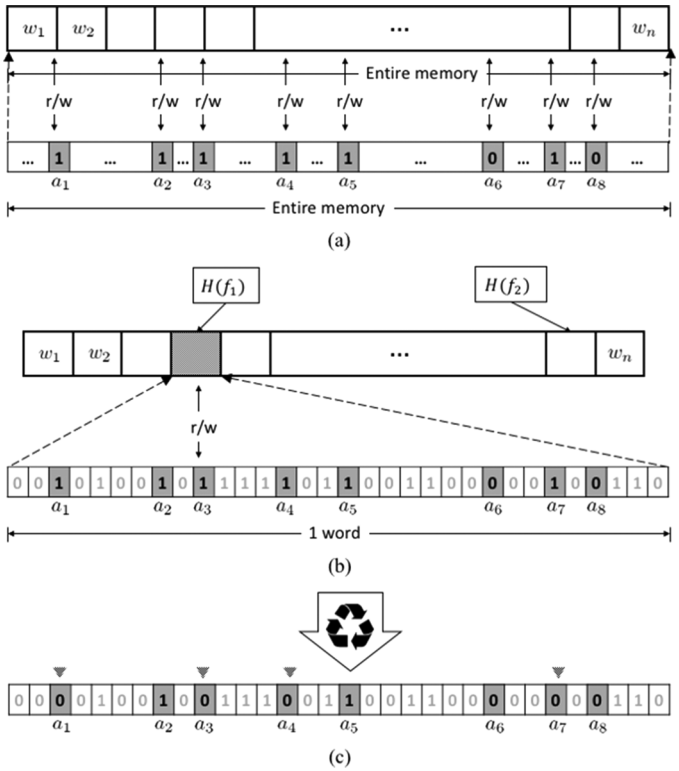

Fig. 1. RCC's Confinement and Recycling. (a) A virtual vector with no confinement (shaded) is dispersed over the entire memory space. 8 memory reads and writes (r/w) are required. (b) A virtual vector with confinement (shaded) is over one word block. Now, it becomes almost (more than 70%) full. 1 memory r/w is required. (c) The virtual vector (shaded) is now recycled after being reset to the average noise level. To reset, randomly selected bits are set to zero as indicated by triangles (▼).

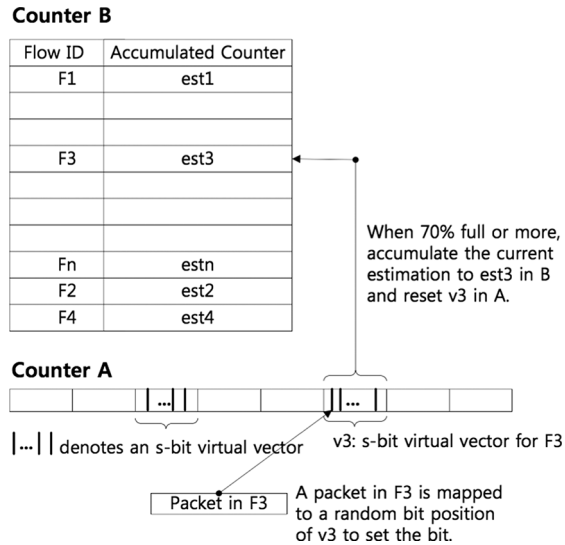

Fig. 2. Recyclable Counter with Confinement.

TABLE I
NOTATION

Notation			
$k(\hat{k})$	actual(estimated) flow size	c	confinement size
c_B	counter field in B	n_A	no. of pkts in counter A
m_A	memory size for A	m_B	no. of slots of counter B
$H_A()$	hash function	f	flow ID
v_f	virtual vector for f	s	virtual vector size
z_f	no. of 0's in v_f	Z	no. of 0's in A
V_s	fraction of 0's in v_f	V_m	fraction of 0's in A

Reminded that a randomized CSE is used for the probabilistic counting structure, the most time-consuming part would be memory accesses. This is because the bits of a virtual vector are dispersed over the entire memory space in CSE, and thus, each and every word holding the bits must be read into a processor to determine whether it is 70% full or not. In addition, when the average noise level should be written to a virtual vector that consists of multiple bits dispersed over the entire memory space, just as many multiple read/writes of memory blocks should be performed, as shown in Fig. 1(a). Consequently, the number of memory accesses is proportional to the size of the virtual vector. Although we use a very small virtual vector size s, such as $s = 8 - 32$, its encoding needs 8–32 memory block read/writes.

To reduce the number of memory accesses, instead of determining bit positions of a virtual vector over the entire memory space, we confine a virtual vector of RCC to one memory block so that it requires only one memory block access to read and write a virtual vector. Fig. 1 shows the idea of confinement and recycling in RCC. Fig. 1(a) is when there is no confinement, where a virtual vector is distributed over the entire memory space, whereas in Fig. 1(b), a virtual vector is confined in one word memory block. In terms of speed, the no-confinement case, Fig. 1(a), requires s number of memory accesses to form a virtual vector, whereas RCC with confinement, Fig. 1(b), requires only one memory access, irrespective of the virtual vector size s. Fig. 1(c) depicts the recycling of RCC described in Section III-A.

Although the confinement reduces the accuracy slightly, the effect is not significant, and the advantage in speed outweighs the loss in accuracy. The larger the confinement size, the better the accuracy, which will be shown in Section VI. Therefore, if routers having a wider bus appear, RCC will work better owing to the lessened confinement effect. In the experiment in Section VI, we conservatively assumed a narrow confinement (32-bit) for fair comparison.

IV. Recyclable Counter With Confinement

In this section, we describe our per-flow measurement algorithm, which consists of two parts: the encoding function, which increases the counters of each flow for incoming packets, and the decoding function, which returns the estimated flow size. See Fig. 2 to understand the two data structures of RCC. Table I contains notation used.

A. Data Structures

RCC has two types of data structure, one for the probabilistic counter (A, the randomized counter) and one for the deterministic accumulator (B, the hash table).

The randomized counter, designed by modifying CSE [23], stores the sizes of all flows into a bit vector, the size of which is m_A bits. All bits are initialized to 0; s randomly-chosen bits are allocated to each flow, and therefore, some flows may share some bits. However, because of the confinement

of the virtual vector, the s bits of the virtual vector are not scattered over the entire memory space but are confined to a c-bit word. To form a virtual vector v_f, a one memory block (or a word) is chosen randomly, and then s bits are selected randomly within the block. To do so, we define hash function $H_A(f) \rightarrow [w, a_1, a_2, \ldots, a_s]$, which randomly outputs an index for the block that corresponds to the flow, f, and v_f, a vector consisting of s indices within the block for the bits of the virtual vector. The bit length of w is $\log_2(m_A/c)$, and that of a_i $(i = 1, \ldots, s)$ is $5 = \log_2(32)$ for the 32-bit confinement. In total, the output of $H_A()$ is $(\log_2(m_A/c) + 5 \cdot s)$ bits long. For s, 8 to 32 bits are sufficient, which is in contrast to 100 to 1,000 bits with CSE [23]. Therefore, with 16 Mb (= 2 MB) of memory in our experiment, 57 $(= \log_2(4\,\text{Mb}/32) + 5 \times 8)$ bits of a hash value are sufficient when the confinement size is 32. Network processors normally have an internal hardware hash unit to output 48, 64, and 128 bits, so one hash computation can generate $[w, a_1, a_2, \ldots, a_s]$ [5].

The hash table is an m_B-size array C_B of (f, c_B) pairs, where f is a b_f-bit flow ID, and c_B is a b_B-bit cumulative counter. All entries are initialized to 0. To resolve the hash table collision, we use open addressing with quadratic probing. That is, to find an empty bucket for a new flow f for encoding, we calculate the hash index with $(H_B(f)+i^2) \bmod m_B$ starting with $i = 1$, and increasing i by 1 until an empty bucket is found. When an empty bucket is found, the flow ID is copied to $C_B[(H_B(f)+i^2) \bmod m_B]$'s f field, and the quantity in the randomized counter is added to $C_B[(H_B(f)+i^2) \bmod m_B]$'s c_B field, where $H_B()$ is a random uniform hash function, the range of which is $[1, m_B]$. Lookup is performed using the same hash until either a match or an empty bucket is found (the latter means "not exist"). It is well known that the expected number of trials is less than $1/(1 - n/m_B)$ with open addressing, where n is the number of elements, and m_B is the number of buckets. When the load factor ($lf = n/m_B$) is 50%, the expected number of trials is less than 2 [10].

B. Encoding

When a packet arrives from flow f, our encoder computes $H_A(f)$ to obtain the indices of the corresponding word for f in memory (w), and the bit positions within the word $(a_1, \ldots, a_s)$. Using the index for a word (w), the instruction read() reads a c-bit word from memory into the c-bit register (Reg), among which s-bits form the virtual vector for f. From then on, all the operations are run on Reg until the write() instruction writes Reg into memory, where read() and write() instructions need one memory block access. The positions of bits constituting the vector on Reg are determined by $(a_1, \ldots, a_s)$, where $1 \le a_i \le c$. In particular, when $s = 32$ and $c = 32$, all the bits in the register are used by a virtual vector, so $(a_1, \ldots, a_s)$ is not necessary. A random number is generated by rand(), which ranges from 1 to s. The encoder chooses a random bit of the virtual vector on Reg to set it to 1, and decrements Z (the total number of 0's in the entire memory) by 1, but only if the bit is 0. To find the number of packets in the vector, z_f, the number of 0's in the vector must be known, and NofZ$(Reg, a_1, \ldots, a_s)$ returns z_f by examining the s number of bits in $Reg[a_1, \ldots, a_s]$. Using the divide and conquer strategy, the number of steps for NofZ() for an s-bit vector is only $\log_2(s)+1$ steps using AND, SHIFT, and ADD instructions [9]. Considering that s is between 8 and 32, it takes only 4 to 6 steps to find the number of 0's.

Linear counting works well until the number of 0's makes up only 30% of a vector [12]. Thus, if the number of 0's in the virtual vector remains less than 30% of the virtual vector, it accumulates the estimated count (est_f) to counter B, and resets some bits of the virtual vector so that the number of 0's in the vector is in proportion to the average number of 0's over the entire memory $(= Z/m_A)$. SetZ$(Reg, a_1, \ldots, a_s, s \cdot Z/m_A - z_f)$ conducts this resetting by flipping the bits holding 1 in $Reg[a_1, \ldots, a_s]$, where the number of bits to be flipped is $s \times Z/m_A - z_f$. Then, Z should be kept up to date by adding the number of flips to calculate the average noise level, which is used to determine whether the virtual vector should be reset or not, and for calculating the number of packets counted by the vector. Finally, the value in Reg is written into the w^{th} word in the memory by the write() instruction.

Because the number of flows is large and some of the flows are large in size, a virtual vector will be shared by some flows. Z can be viewed as a random variable holding the number of 0's over the entire memory. The total number of distinct packets during a measurement period is estimated as $\hat{k_1} = m_A \ln(m_A/Z)$, since $E(Z) \approx m_A e^{-k_2/m_A}$. For more detail, refer to Appendix A in Whang *et al.* [12]. Therefore, on average, one virtual vector has $\hat{k_1} \times s/m_A = s \ln(m_A/Z)$ packets that are contributed to by flows other than f [23]. Because of this noise, the virtual vector should be reset to $s \ln(m_A/Z)$ instead of zero, which preserves the information contributed by the small flows. Similarly, the number of distinct packets in the virtual vector is $\hat{k_2} = s \ln(s/z_f)$. Thus, the net number of distinct packets contributed by f is $\hat{k} = s \ln(s/z_f) - s \ln(m_A/Z)$[23].

However, instead of directly following Yoon *et al.* [23], we use a more precise estimation, because the intermediate results are accumulated many times to a counter in the hash table. The accumulation of many small errors in each intermediate estimation might result in a large error. Let A_j be an event where the j-th bit is 0, and let 1_{A_j} be its indicator random variable. Then, the number of bits with 0 is $U_n = \sum_{j=1}^{m_A} 1_{A_j}$. From Appendix A in Whang *et al.* [12],

$$E(U_n) = \sum_{j=1}^{m} P(A_j) = m_A \left(1 - \frac{1}{m_A}\right)^n$$

where $P(A_j)$ is the probability of A_j, and thus, $P(A_j) = (1 - 1/m_A)^n$. Let V_m and V_s be the fraction of 0's in the entire memory and in the virtual vector, respectively. By replacing $\frac{E(U_n)}{m_A}$ with our measurement V_m, we obtain

$$V_m = \left(1 - \frac{1}{m_A}\right)^{\hat{n}}$$

and by taking the log

$$\ln(V_m) = \hat{n} \cdot \ln\left(1 - \frac{1}{m_A}\right).$$

Thus, the estimation is

$$\hat{n} = \frac{\ln(V_m)}{\ln(1 - 1/m_A)}. \tag{2}$$

Algorithm 1: Encode(f)

$(w, a_1, \ldots, a_s) \leftarrow H_A(f)$
$Reg[1, \cdots, c] \leftarrow$ **read**(w)
$v \leftarrow$ rand()
if $Reg[a_v] = 0$ **then**
  $Reg[a_v] \leftarrow 1$
  $Z \leftarrow Z - 1$
$z_f \leftarrow$NofZ$(Reg, a_1, \ldots, a_s)$
if $z_f < 0.3 \times s$ **then**
  if $z_f = s$ **then**
    $est_f \leftarrow s \ln(s) + \gamma \cdot s + 0.5 - \frac{s \cdot \ln(Z/m_A)}{m_A \cdot \ln(1-1/m_A)}$
  else
    $est_f \leftarrow \frac{\ln(z_f/s)}{\ln(1-1/s)} - \frac{s \cdot \ln(Z/m_A)}{m_A \cdot \ln(1-1/m_A)}$
  SetZ$(Reg, a_1, \ldots, a_s, s \times Z/m_A - z_f)$
  $Z \leftarrow Z + s \times Z/m_A - z_f$
  Acc$_B(f, est_f)$
if *Reg* has been changed, **then**
  write(w, Reg)

Fig. 3. Encoding Algorithm.

Instead of using $-m_A \ln(V_m)$ that is obtained by approximating $(1 - 1/m_A)^{\hat{n}} \approx e^{-\hat{n}/m_A}$, we are going to use (2) for estimation. This reduces approximation error slightly, which is important in RCC, because small approximation errors are accumulated many times for large flows, which results in a large estimation error. With CSE, this does not make a big difference, because there is no repeated accumulation of estimated values. To sum up, the estimation for $\hat{k}$ is

$$\hat{k} = \frac{\ln(V_s)}{\ln(1 - 1/s)} - \frac{s \cdot \ln(V_m)}{m_A \cdot \ln(1 - 1/m_A)}. \quad (3)$$

When the vector is full of 1's, however, the formula cannot be applied because z_f is 0, which yields infinity for $\ln(s/z_f) = \ln(V_s)$. In this case, to calculate $\hat{k_2}$, we take advantage of the coupon collector's problem, where the solution is

$$E(k_2) \approx s \ln(s) + \gamma \cdot s + 0.5 \quad (4)$$

where $\gamma \approx 0.5772156649$ is the Euler-Mascheroni constant. Algorithm 1 in Fig. 3 summarizes this, where Acc$_B(f, est_f)$ is the function that locates the bucket for flow f by hashing with quadratic probing, and then increments the bucket by est_f. Here, Z can be computed in a register inside the CPU, which removes memory access for updating Z as in CSE.

C. Decoding

The estimation of the given flow f is very simple as shown in Fig. 4. The count in the randomized counter and that in the hash table for the flow are summed. However, if the virtual vector is full of 1's, we estimate the count by the coupon collector's problem instead of CSE. Therefore, the estimated flow size is as follows: When $s \neq z_f$,

$$\hat{k} = c_B + \frac{\ln(z_f/s)}{\ln(1 - 1/s)} - \frac{s \cdot \ln(Z/m_A)}{m_A \cdot \ln(1 - 1/m_A)}. \quad (5)$$

When $s = z_f$,

$$\hat{k} = c_B + s \ln(s) + \gamma \cdot s + 0.5 - \frac{s \cdot \ln(Z/m_A)}{m_A \cdot \ln(1 - 1/m_A)} \quad (6)$$

z_f, the number of 0's in the virtual vector for f, is obtained by NofZ$(Reg, a_1, \ldots, a_s)$, and c_B is a counting value for f in the hash table retrieved by calling Count$_B(f)$. If f exists in the table, c_B has an accumulated value; otherwise it is equal to zero.

Algorithm 2: Decode(f)

$est \leftarrow$ Count$_B(f)$
$(w, a_1, \ldots, a_s) \leftarrow H_A(f)$
$Reg[1, \cdots, c] \leftarrow$ **read**(w)
$z_f \leftarrow$ NofZ$(Reg, a_1, \ldots, a_s)$
if $z_f = s$ **then**
  $est \leftarrow est + s \ln(s) + \gamma \cdot s + 0.5 - \frac{s \cdot \ln(Z/m_A)}{m_A \cdot \ln(1-1/m_A)}$
else
  $est \leftarrow est + \frac{\ln(z_f/s)}{\ln(1-1/s)} - \frac{s \cdot \ln(Z/m_A)}{m_A \cdot \ln(1-1/m_A)}$
return est

Fig. 4. Decoding Algorithm.

For decoding, flow IDs in the hash table allow us to retrieve the list of heavy uploaders/downloaders directly, whereas most existing schemes do not support this functionality without the addition of more memory for flow labels.

V. Estimation Accuracy

In this section, we analyze the accuracy of RCC in terms of expected estimation accuracy and variance.

CSE is for counting distinct elements, but by randomizing the destination address, we can obtain a flow measurement algorithm, the behavior of which, with regard to mean and variance, is the same as CSE. Thus, the accuracy analysis broadly follows that found in Yoon *et al.* [23]. The difference comes from both the smaller s of RCC, compared to CSE, and the accumulation of the small estimation results. Therefore, we examine the implication of the two changes in terms of the mean and the variance of $\hat{k}$. Here, we note that we are going to use CSE and linear counting's estimation, which does not give us an advantage in the analysis, because our estimation in (3) is more accurate than that of CSE shown in (1).

Theorem 1: Assuming that the hash table has enough space to store overflown flows, $m_A = O(n_A)$, where n_A is the total number of packets held by the randomized counter, m_A is the total number of bits of the randomized counter, $\hat{k_{RCC}}$, the estimation of k by RCC is unbiased, and it satisfies

$$Var(\hat{k}_{RCC}) < Var(\hat{k}_{CSE})$$

where $\hat{k}_{CSE}$ is a random variable that has counts for a flow estimated by CSE.

Proof: The proof is found in four cases: a small flow that is not reset and a large flow that has been reset multiple times. For each, expectation and variance of $\hat{k}$ will be examined. If $k_1 = -s \cdot \ln(E(V_m))$ and $k_2 = -s \cdot \ln(E(V_s))$, then $k \simeq -k_1 + k_2$. Let $\alpha = (n_A/m_A) + (k/s)$.

Case 1: First, let us see $E(\hat{k})$ for a small flow. $E(\hat{k_1})$ and $E(\hat{k_2})$ are from Yoon *et al.* [23].

$$E(\hat{k_1}) \simeq \frac{s}{m_A}\left(n_A + \frac{e^{n_A/m_A} - (n_A/m_A) - 1}{2}\right)$$
$$= \frac{s \cdot n_A}{m_A} + \frac{s \cdot (e^{n_A/m_A} - (n_A/m_A) - 1)}{2m_A}. \quad (7)$$

When $m_A \simeq n_A$, $E(\hat{k_1}) \simeq s(n_A/m_A)$. The second term of $E(\hat{k_1})$ is the approximation error, for which less is better.

Because of the smaller s of RCC, the approximation error for $E(\hat{k_1})$ is smaller. For k_2,

$$E(\hat{k_2}) = s \cdot (\alpha + \frac{e^{\alpha} - 1 - (k/s)}{2s}) = \alpha s + \frac{e^{\alpha} - 1 - (k/s)}{2}. \quad (8)$$

The second term of $E(\hat{k_2})$ is the approximation error, which is

$$\frac{e^{(n_A/m_A)+(k/s)} - 1 - (k/s)}{2}. \quad (9)$$

Because the maximum flow size that is countable by a virtual vector is bounded by $s \cdot \ln(s) + \gamma \cdot s + 0.5$, the approximation error of the largest flow size is proportional to $k/s \simeq \ln(s) + \gamma$.

$$\frac{e^{(n_A/m_A)+\gamma} \cdot s - 1 - (\ln(s) + \gamma)}{2}. \quad (10)$$

Therefore, s is smaller, which is better. Consequently, with better approximation than CSE using a larger s, we can obtain $E(\hat{k_2}) \simeq s\alpha = s(n_A/m_A) + k$. Hence,

$$E(\hat{k}) = -E(\hat{k_1}) + E(\hat{k_2}) \simeq k \quad (11)$$

which means the mean of $\hat{k}$ is close to k.

Case 2: Now, we consider the variance of $\hat{k}$ for a small flow.

$$Var(\hat{k}) = Var(\hat{k_1}) + Var(\hat{k_2}) - 2Cov(\hat{k_1}, \hat{k_2}) = Var(\hat{k_1}) + Var(\hat{k_2}) + 2(E(\hat{k_1})E(\hat{k_2}) - E(\hat{k_1}\hat{k_2})),$$

where variances of $\hat{k_1}$ and $\hat{k_1}$ are

$$Var(\hat{k_1}) \simeq \frac{s^2}{m_A}(e^{n_A/m_A} - \frac{n_A}{m_A} - 1)$$
$$Var(\hat{k_2}) \simeq s(e^{\alpha} - \frac{k}{s} - 1).$$

$E(\hat{k_1}\hat{k_2})$ is approximated by

$$E(\hat{k_1}\hat{k_2}) \simeq \frac{s^2 n_A \alpha}{m_A} + \frac{n_A s(e^{\alpha} - 1 - \frac{k}{s})}{2m_A} + \frac{s^2\alpha(e^{n_A/m_A} - \frac{n_A}{m_A} - 1)}{2m_A}.$$

Therefore, variance of $\hat{k}$ is

$$Var(\hat{k}) = s(e^{\alpha} - \frac{k}{s} - 1) + \frac{s^2}{m_A}(e^{n_A/m_A} - \frac{n_A}{m_A} - 1) + \frac{s \cdot (e^{n_A/m_A} - n_A/m_A - 1)(e^{\alpha} - 1 - k/s)}{2m_A}. \quad (12)$$

In (12), the first term is the dominating factor, and the second and the third terms can be ignored, because $(e^{n_A/m_A} - \frac{n_A}{m_A} - 1)$ in the second term is small when $n_A \simeq m_A$, and m_A is large compared with s in the third term. Now, it is clear that the smaller s is, the less the variance of $\hat{k}$. In the case of a small flow, it is certain from (12) that $Var(\hat{k}_{RCC})$ is smaller than $Var(\hat{k}_{CSE})$. For example, with $s = 128$ for CSE, the first term of $Var(\hat{k})$ is about 44,000, because α is about 5.85 assuming $m_A \simeq n$ when the virtual vector is full of 1's. When $s = 16$ for RCC, however, it is only about 700. For small flows, the following is satisfied:

$$Var(\hat{k}_{RCC}) < Var(\hat{k}_{CSE}). \quad (13)$$

We have proven the arguments on the expectation and the variance of $\hat{k}$ for a small flow. Now, let us see if the arguments hold for a large flow.

Case 3: We show that $E(\hat{k}_{RCC})$ for a large flow that causes a reset, and is registered in the hash table, is unbiased. $E(\hat{k}_{RCC})$ for a small flow that has not overflowed is not biased according to (11). Let r be the reset number for a large flow. Because $H_A()$ has a uniform and random distribution, $E(r) \approx \frac{k}{s \ln(s)}$. Let k_i be the portion of the actual flow size k at the moment of the i^{th} reset, and let $\hat{k}^i_{RCC}$ be the estimation of k_i. Thus, the actual flow size $k = \sum_{i=1}^{r} k_i$.

$$E(\hat{k}_{RCC}) = E(\sum_{i=1}^{r} \hat{k}^i_{RCC}) = \sum_{i=1}^{r} E(\hat{k}^i_{RCC}). \quad (14)$$

The only difference between a case where the number is reset and one where the counter has not overflowed is where the segment of the flow reset is counted on top of the average noise level ($= s \ln(m_A/Z)$). Therefore, we can say that only the capacity of the counter decreased somewhat in the resetting. Thus, the analysis in (11) is still valid in the resetting, and therefore, $E(\hat{k}^i_{RCC}) = k_i$. Consequently,

$$E(\hat{k}_{RCC}) = \sum_{i=1}^{r} E(\hat{k}^i_{RCC}) = \sum_{i=1}^{r} k_i = k \quad (15)$$

which means $E(\hat{k})$ is unbiased.

Case 4: Now, we show that for a large flow, $Var(\hat{k}_{RCC})$ is always smaller than $Var(\hat{k}_{CSE})$. For a better explanation of the variance for large flows that are accumulated in the hash table, $\hat{k}_{CSE}$ is broken into r pieces, $\hat{k}^1_{CSE}, \ldots \hat{k}^r_{CSE}$, where $\hat{k}^i_{CSE}$ is an estimation incremented from the $(i-1)^{th}$ to the i^{th} measurement period that is calculated from (1). Let z^i_f be the number of 0's at the $(i-1)^{th}$ measurement period in the virtual vector for f, and let Z^i be that in the entire memory during the same period. $\hat{k}^i_{CSE}$, $\hat{k_2}^i_{CSE}$, and $\hat{k_1}^i_{CSE}$ can be viewed as the independent random variables as $\hat{k}_{CSE}$, $\hat{k_2}_{CSE}$, and $\hat{k_1}_{CSE}$. Therefore, $\hat{k}^i_{CSE} = \hat{k_2}^i_{CSE} - \hat{k_1}^i_{CSE}$ because z^i_f and Z_i are the number of 0's in the virtual vector of f and the entire memory at the i^{th} measurement period, respectively,

$$\sum_{j=1}^{i} \hat{k_2}^j_{CSE} = s \ln(s/z^i_f),$$

and

$$\sum_{j=1}^{i} \hat{k_1}^j_{CSE} = s \ln(m_A/Z^i).$$

Then,

$$\sum_{i=1}^{r} \hat{k}^i_{CSE} = \sum_{i=1}^{r} (\hat{k_2}^i_{CSE} - \hat{k_1}^i_{CSE}) = \sum_{i=1}^{r} \left[\sum_{j=1}^{i} (\hat{k_2}^j_{CSE}) - \sum_{j=1}^{i-1} (\hat{k_2}^j_{CSE}) \right] - \sum_{i=1}^{r} \left[\sum_{j=1}^{i} (\hat{k_1}^j_{CSE}) - \sum_{j=1}^{i-1} (\hat{k_1}^j_{CSE}) \right]$$

$$
\begin{aligned}
&= \sum_{i=1}^{r}\left[\sum_{j=1}^{i}(\hat{k_2}^{j}_{CSE}) - \sum_{j=1}^{i}(\hat{k_1}^{j}_{CSE})\right] \\
&\quad - \sum_{i=1}^{r}\left[\sum_{j=1}^{i-1}(\hat{k_2}^{j}_{CSE}) - \sum_{j=1}^{i-1}(\hat{k_1}^{j}_{CSE})\right] \\
&= \sum_{i=1}^{r}[s\ln(s/z_f^i) - s\ln(m_A/Z^i)] \\
&\quad - \sum_{i=1}^{r}[s\ln(s/z_f^{i-1}) - s\ln(m_A/Z^{i-1})] \\
&= [s\ln(s/z_f^r) - s\ln(m_A/Z^r)] \\
&\quad - [s\ln(s/z_f^0) - s\ln(m_A/Z^0)] \\
&= s\ln(s/z_f) - s\ln(m_A/Z) = \hat{k}_{CSE},
\end{aligned}
$$

where $[s\ln(s/z_f^0) - s\ln(m_A/Z^0)] = 0$, because there is no packet at the beginning of a measurement period. Thus, the summation of the broken pieces perfectly recovers $\hat{k}_{CSE}$. Because $\hat{k}^i_{CSE}$s are independent of each other, $Var(\hat{k}_{CSE}) = Var(\sum_{i=1}^{r}\hat{k}^i_{CSE}) = \sum_{i=1}^{r}Var(\hat{k}^i_{CSE})$. Similarly, $Var(\hat{k}_{RCC}) = \sum_{i=1}^{r}Var(\hat{k}^i_{RCC})$. Because each s^i_{CSE} segment is so small as to not be registered in the hash table, we can apply (13), and therefore, if we compare $\hat{k}^i_{CSE}$ with $\hat{k}^i_{RCC}$, $Var(\hat{k}^i_{RCC}) < Var(\hat{k}^i_{CSE})$ for $i = 1 \ldots r$. Thus, we can also conclude for flows registered in the hash table that

$$
\begin{aligned}
Var(\hat{k}_{RCC}) &= \sum_{i=1}^{r}Var(\hat{k}^i_{RCC}) \\
&< \sum_{i=1}^{r}Var(\hat{k}^i_{CSE}) = Var(\hat{k}_{CSE}).
\end{aligned}
$$

This completes the proof.

In summary, the mean of our estimation results is approximately the same as the mean of the actual flow sizes, and the variance in ours is always smaller than that of CSE, because of the smaller virtual vector size s.

VI. Experiment

In this section, we evaluate the performance of our algorithm in terms of processing time and estimation accuracy, and compare RCC with recently published schemes such as PMC [22] and CSM [24]. A flow is defined as per-source IP, and the size of a flow is the number of packets in the flow for presentation. We note here that a flow also can be defined as various different types, such as per-source IPs, per-destination IPs, and per-source/destination IP pairs.

A one-minute network traffic trace from the CAIDA dataset [4] was used, which had 40.5 million packets where the size is 43 GB from an OC-192 link (thus, $\simeq$ 5.3 Gbps) holding 264 K flows. The flow size ranged from 1 to 3,327,267 packets. The traffic was collected at the Equinix Chicago data center from 13:10-13:11 on the 21st of November, 2013. See Fig. 5 for the traffic distribution of the dataset. The total memory allocated for the measurements varied from 8 Mb (= 1 MB) to 16 Mb (= 2 MB). In each measurement period, 40.5 million packets were processed. On an OC-192 link (10+ Gbps), this amounts to a 30-second measurement period, and on an OC-768 link (40+ Gbps), an eight-second measurement period assuming an average packet size of 1,000 bytes and 100% link utilization. The measurement period should be adjusted considering memory available and the amount of data to be processed, where the amount of data is dependent upon the data rate.

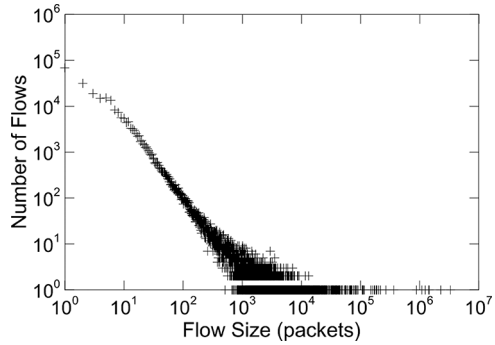

Fig. 5. Traffic distribution of the CAIDA dataset. Each point represents the number of flows that reach a certain size in packets.

A. Parameters

Parameters in RCC should be analyzed to achieve the best accuracy given the memory constraints and encoding/decoding speed. The number of bits, the confinement size, and the size of the virtual vector for the randomized counter determine the accuracy of the counter, and the number of entries in the hash table and the size of each entry determine the size of the hash table.

To make RCC applicable to old network processors, confinement size c was chosen as 32 bits, which gave us less accuracy than when a large c is chosen. Thus, the experiment result for RCC in this section is conservative and can be improved by increasing c to 128 or more, which is shown in Fig. 13.

The maximum size of a flow in the randomized counter is determined by $s\ln(s) + \gamma \cdot s + 0.5$. This maximum allowable flow size for a virtual vector sets the threshold for a large flow, and the number of flows larger than this threshold in the given traffic decides the number of entries (m_{top}) in the hash table. Thus, given a typical traffic distribution on a site, it is possible to choose the size of the hash table by adjusting s as we decide the other parameters according to the length of a measurement period. In the experiment, between 8 to 32 was sufficient for s in RCC, and 16–34% of total flows were registered in the hash table. m_B denotes the number of pairs in the hash table, which was set equal to $m_{top} \times 2$, considering the load factor (lf) of 50%. Let us denote as b_B the bit length of a counter (c_B) in the hash table, which sets the maximum allowable number of packets in a flow to $2^{b_B} - 1$. We define a flow by its source IP address, the length (b_f) of which is 32 bits, and $b_B = 22$ bits, which can count up to $2^{22} - 1 \simeq 4$ M packets (or 4 GB assuming a packet is 1,000 bytes long) for each flow. Thus, in total, $(32 + 22) \times m_B$ bits (denoted as M_B) are allocated to the hash table. Given the available amount of memory M, the total memory for the randomized counter is $m_A = M - M_B$ bits.

For PMC, we used parameters selected according to Lieven and Scheuermann [22], and for CSM, the size of the virtual

TABLE II
PARAMETERS FOR THE EXPERIMENT

RCC					
Param	8Mb	16Mb	Param	8Mb	16Mb
M_B	4.5Mb	12Mb	b_B	22	22
m_B	87K	233K	b_f	32	32
m_{top}	43K	90K	m_A	3.5Mb	4.0Mb
lf	49%	39%	s	32	8
CSM			PMC		
Param	8Mb	16Mb	Param	8Mb	16Mb
m	8Mb	16Mb	N	8Mb	16Mb
b	8	8	w	32	32
l	10,000	10,000	m	256	256

TABLE III
NUMBER OF OPERATIONS PER PACKET FOR ENCODING

	Mem. Accesses	Hashes (Clocks)
RCC	$\geq$1	$\geq$1 ($\approx$30)
PMC	1	1 ($\approx$30)
CSM/MLM	1	1 ($\approx$30)
MRSCBF	4.47	4.47 ($\approx$134)
CB	$\geq$3	$\geq$3 ($\approx$90)

vector was chosen to be large enough to count the maximum flow size ($l = 10,000$). The parameters are listed in Table II.

B. Processing Speed

The processing time was evaluated according to the amount of time required for encoding and decoding. In general, it depends on the number of memory accesses and hash function evaluations. The numbers of each operation in each scheme are summarized in Table III, where the number of clock cycles required is obtained by assuming an Intel's CRC-based hash function that requires less than 30 cycles/hash [7].

In terms of the number of memory accesses and hash computations for encoding, PMC, CSM and MLM, which require only one read/write and one hashing, are better than the other algorithms [22], [24]. CB requires 3+ memory accesses and 3+ hash computations, where + represents counter overflow. When the counter overflows, the second layer should be updated, which requires another three memory accesses and three hashes. MRSCBF, which has $(a_1 = 3, a_2 = 4, a_3 = 6, a_4 = 6, \ldots, a_9 = 6)$ hashes and $(p_1 = 1, p_2 = 1/4, p_3 = 1/4^2, \ldots, p_9 = 1/4^8)$ probabilities for 9 filters, respectively, requires on average $\sum_{i=1}^{9} p_i \cdot a_i = 4.47$ writes and 4.47 hash computations.

RCC is comparable to CSM/MLM. It requires one memory access and one hash computation for every packet. When RCC should accumulate and reset a vector, one read (that is, one 54-bit memory block read to find the bucket and its counter), one write and one hash computation are demanded additionally. The reset occurs with a probability that is dependent upon m_A, s, and the input traffic distribution. According to our experiment, about 6.6% (for a 32-bit virtual vector) to 21% (for an eight-bit virtual vector) of the total packets caused the resetting, which can be further reduced by increasing m_A or s, if necessary. The average number of memory accesses is 1.07 ($= 1 + 0.066$) to 1.21 ($= 1 + 0.21$), and that of hash computations was 1.07 to 1.21. The fraction of total packets that found their slots in the hash table in more than two hash computations was only 0.22% and 0.23% for an eight-bit virtual vector and for a 32-bit virtual vector, respectively. Only one packets out of 40.5 million packets experienced the maximum number of hashes, which is 14 hashes for an eight-bit virtual vector.

TABLE IV
NUMBER OF OPERATIONS PER FLOW FOR DECODING, WHERE $l = 10,000$ FOR CSM/MLM IS THE VECTOR SIZE, f FOR CB IS THE NUMBER OF TOTAL FLOWS, AND FOR PMC, m IS 32 TO 256 AND $w = 32$. MLE IS MAXIMUM LIKELIHOOD ESTIMATION

	Reads	Hashes	Notes
RCC	$\geq$2	$\geq$2	Constant
PMC	$\gg$256	$\gg$256	$\ll m \times w$
CSM	10,000	10,000	Proportional to l
MLM	$\gg$ 10,000	$\gg$ 10,000	Extra MLE
MRSCBF	$\gg$49	$\gg$49	Extra MLE
CB	N/A	N/A	Linear in f

The processing speed for decoding an individual flow was evaluated with the same metrics as those used for encoding (see Table IV). RCC needs to retrieve one counter from the hash table and one virtual vector from the randomized counter. Thus, for small flows, two hash computations and two memory reads (one for reading a virtual vector and one for checking non-existence of the flows in the hash table) are sufficient, but even for most of large flows, two hash computations and two memory reads (one for reading a virtual vector and one for fetching the accumulated counter in the hash table) are sufficient. CSM/MLM needs at least l hash computations and l memory accesses, where l is 100 to 10,000. MLM uses maximum likelihood estimation for counting values obtained from l counters to improve accuracy, but it requires a very high computational overhead for decoding. The total number of memory accesses (or hash computations) for MRSCBF with the nine filters is 49. However, this is only for gathering intermediate data; extra processing for MLE or mean value estimation (MVE) should be performed for the final estimation. On the other hand, PMC is a hybrid algorithm that has two different estimation algorithms, one for small flows and one for large flows. In the best case scenario (estimating a small flow), it uses m hash computations and memory accesses to get values from m sketches, where $m = 256$ in our experiment. If a flow is large, the number of hashes and memory accesses is from $2 \times m$ to $w \times m$, where w is 32.

As can be seen in Table IV, RCC is significantly faster than the other schemes in terms of decoding speed. Online decoding will lead to the use of per-flow measurements as a new service in areas such as real-time detection of heavy downloaders/uploaders, spammers and DoS attackers.

Finally, Fig. 6 shows a timing diagram when our algorithm is executed on a system with multiple network processors. The x-axis shows the estimation of time in cycles to perform our encoding algorithm for four cases. We estimated the required number of cycles referring to [19]. In our encoding algorithm, there is only one variable shared by multiple network processors, which is Z and it cannot be accessed concurrently by

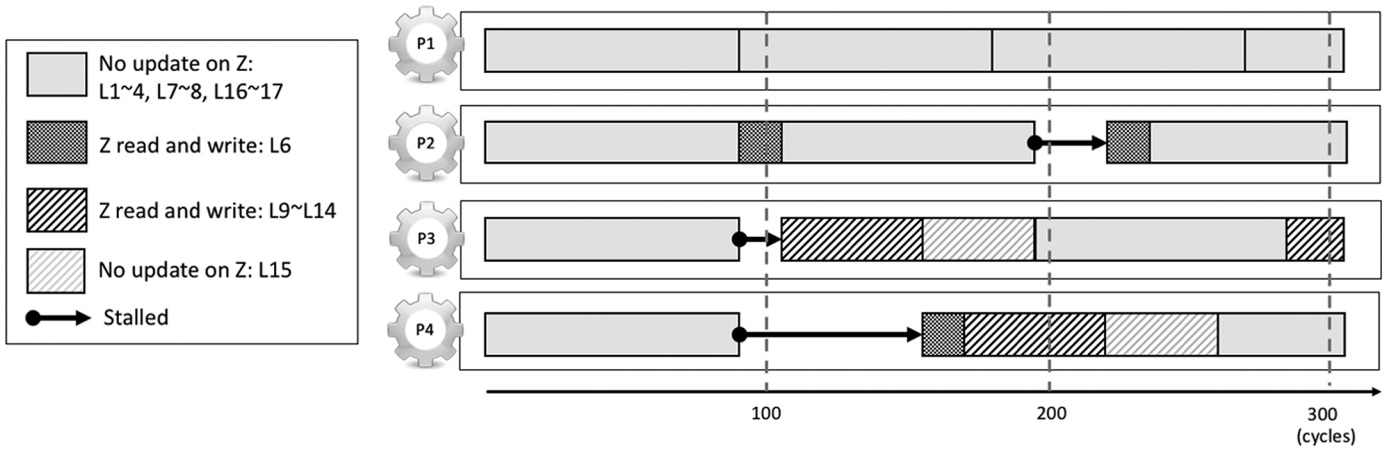

Fig. 6. RCC's timing diagram on a system with multiprocessors. L# denotes the line number of the Algorithm 1 in Fig. 3.

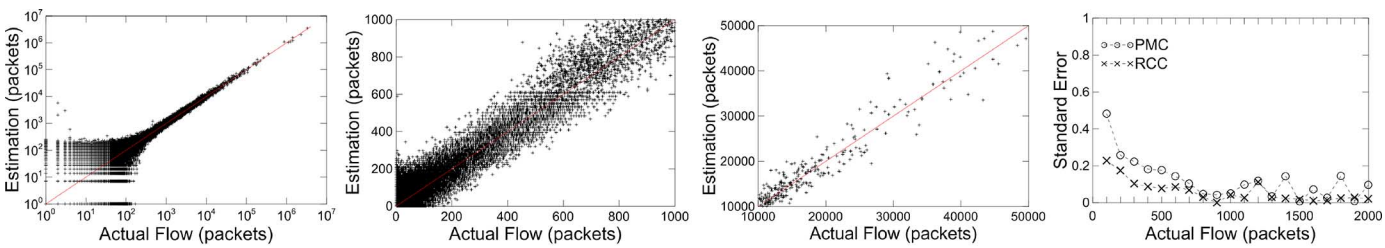

Fig. 7. Estimation result for PMC with 8 Mb (= 1 MB) memory. Each point in (a), (b), and (c) stands for each flow, and the $y = x$ line is the reference line. To see how accurate each algorithm is, check how close every point is to $y = x$. (a) Estimation result in log scale shows the whole estimation range. (b) Estimation in linear scale from 0 to 1 k. (c) Estimation in linear scale from 10 k to 50 k. (d) Standard errors calculated and experimented.

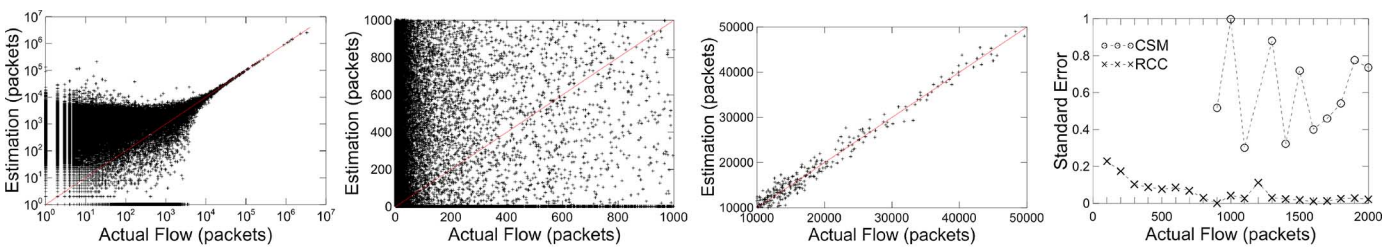

Fig. 8. Estimation result for CSM with 8 Mb (= 1 MB) memory. Each point in (a), (b), and (c) stands for each flow, and the $y = x$ line is the reference line. To see how accurate each algorithm is, check how close every point is to $y = x$. (a) Estimation result in log scale shows the whole estimation range. (b) Estimation in linear scale from 0 to 1 k. (c) Estimation in linear scale from 10 k to 50 k. (d) Standard errors calculated and experimented.

multiple network processors. Therefore, a processor cannot modify the variable Z before another processor completes a read/write interval. To examine more specifically the behavior of our encoding algorithm, we divided our algorithm into four segments: two Z read/write segments (pattern-filled: line 6 and lines 9–14), and the remaining segments (solid gray: lines 1–4, lines 7–8, line 15, and lines 16–17) that do not have any Z read/write instruction. Line 6 requires one instruction for decrement, and lines 9–14 require one logarithm calculation and several write operations. The solid gray segment needs one hash, one random number generation, and one memory read/write operation. There are four cases in which a packet is processed according to how Z is updated. The first case is when Z is not updated. The second case is when Z is updated by the first "if-statement" (line 4). The third case is when the variable is modified by the second "if" (line 8). The last case is when Z is updated by both "if-statement". Here, a pattern-filled segment cannot be executed concurrently by multiple network processors, but it must be exclusively run by only one processor. This is shown in Fig. 6, where "stalled" phases are for exclusive running of the code segment.

C. Estimation Accuracy

In this section, we present the experiment results on the accuracy of the three algorithms (RCC, CSM, PMC). The accuracy was analyzed using two measures: flow size estimation $\hat{k}$ and standard error ($\sqrt{Var(\hat{k})}/k$).

Figs. 7–9 show experiment results for $\hat{k}$ of each algorithm using 8 Mb memory, and Figs. 10–12 show accuracy using 16 Mb memory. In each figure, the left figure is the whole range view of the estimation accuracy in log scale, the second shows the estimation results ranging from 0 to 1,000, and the third shows from 10 K to 50 K in linear scale. The x-axis represents the actual flow size (k), and the y-axis represents the estimated flow size ($\hat{k}$). Each point represents one flow, and if a point for

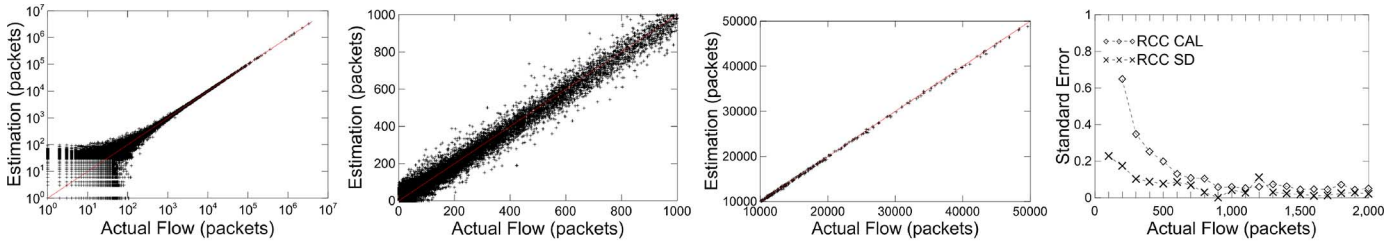

Fig. 9. Estimation result for RCC with 8 Mb (= 1 MB) memory. Each point in (a), (b), and (c) stands for each flow, and the $y = x$ line is the reference line. To see how accurate each algorithm is, check how close every point is to $y = x$. RCC has the top 43,000 source IP addresses in 8 Mb, whereas the others do not have. (a) Estimation result in log scale shows the whole estimation range. (b) Estimation in linear scale from 0 to 1 k. (c) Estimation in linear scale from 10 k to 50 k. (d) Standard errors calculated and experimented.

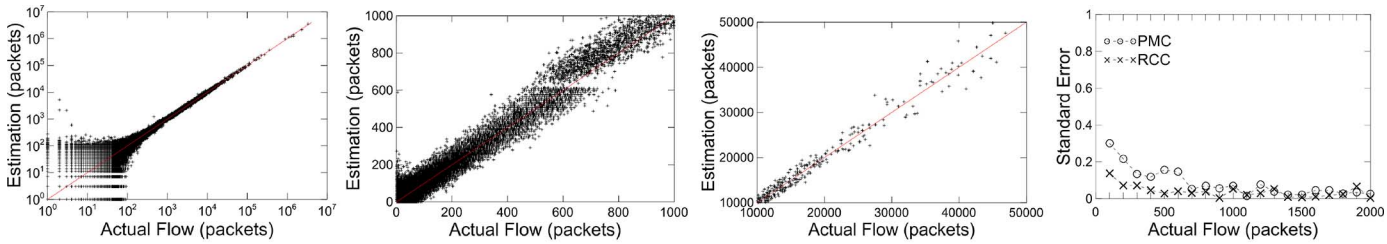

Fig. 10. Estimation result for PMC with 16 Mb (= 2 MB) memory. Each point in (a), (b), and (c) stands for each flow, and the $y = x$ line is the reference line. To see how accurate each algorithm is, check how close every point is to $y = x$. (a) Estimation result in log scale shows the whole estimation range. (b) Estimation in linear scale from 0 to 1 k. (c) Estimation in linear scale from 10 k to 50 k. (d) Standard errors calculated and experimented.

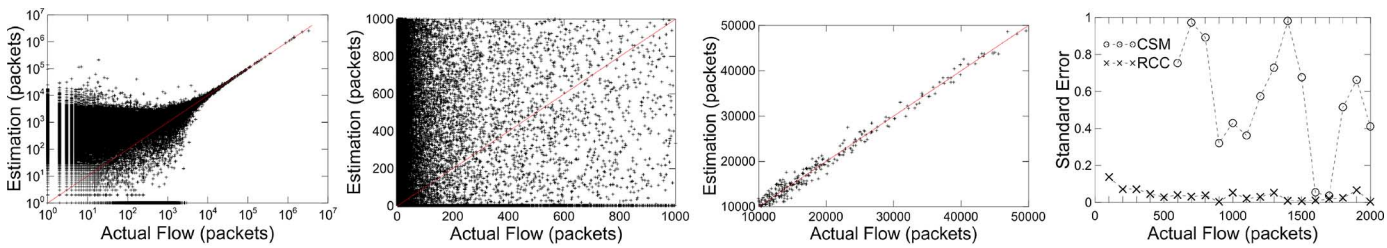

Fig. 11. Estimation result for CSM with 16 Mb (= 2 MB) memory. Each point in (a), (b), and (c) stands for each flow, and the $y = x$ line is the reference line. To see how accurate each algorithm is, check how close every point is to $y = x$. (a) Estimation result in log scale shows the whole estimation range. (b) Estimation in linear scale from 0 to 1 k. (c) Estimation in linear scale from 10 k to 50 k. (d) Standard errors calculated and experimented.

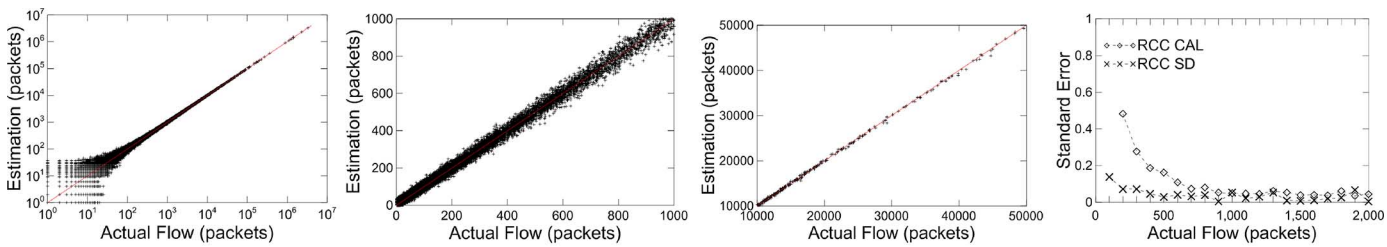

Fig. 12. Estimation result for RCC with 16 Mb (= 2 MB) memory. Each point in (a), (b), and (c) stands for each flow, and the $y = x$ line is the reference line. To see how accurate each algorithm is, check how close every point is to $y = x$. RCC has the top 96,000 source IP addresses in 16 Mb, whereas the others do not have. (a) Estimation result in log scale shows the whole estimation range. (b) Estimation in linear scale from 0 to 1 k. (c) Estimation in linear scale from 10 k to 50 k. (d) Standard errors calculated and experimented.

a flow is exactly on the reference line $y = x$, the estimation is precise. If a point is under the line, it is an under-estimation; otherwise, it is an over-estimation. The rightmost figure depicts $\sqrt{Var(\hat{k})/k}$. Obviously, the more memory an estimator uses, the higher its accuracy.

With 8 Mb of memory, CSM does not work for small flows with sizes (in packets) than 1,000, as shown in Fig. 8(b), and PMC works barely, as shown in Fig. 7(b), whereas RCC works well, as shown in Fig. 9(b). Comparing Figs. 7(c), 8(c) and 9(c), RCC is the best of all, while PMC does not work well.

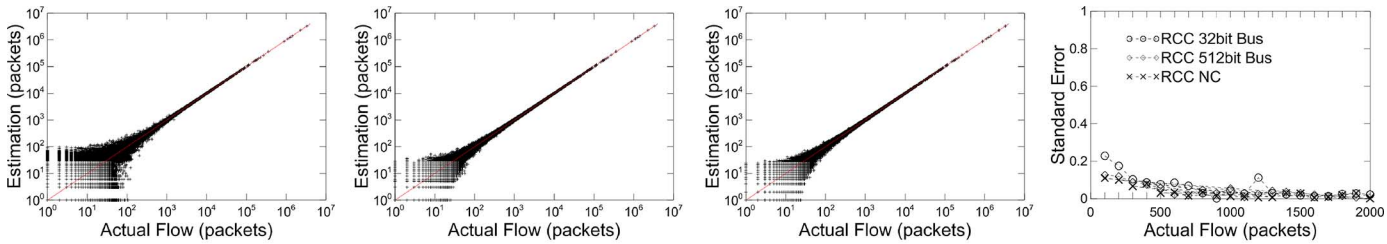

Fig. 13. Estimation result with 8 Mb (= 1 MB) memory according to the confinement size. Each point (in cross) stands for each flow, and the $y = x$ line is the reference line. Both axes are in log scale. To see how accurate each algorithm is, check how close every point is to $y = x$. All have the same virtual vector size (32 bits). RCC NC means that no-confinement was applied. Confining a virtual vector in a larger block results in better estimation. (a) Confinement size $c = 32$. (b) Confinement size $c = 512$. (c) No confinement. (d) Standard errors of (a)–(c).

Figs. 7(d) and 8(d) show that the standard error of RCC is significantly lower than the others, which means RCC's estimation is always tighter than those of CSM and PMC. Figs. 9(d) and 12(d) confirm that calculated standard errors match the experiment results.

With 16 Mb of memory, PMC's estimation for small flows is better than that of CSM, which is yet not meaningful (See Figs. 10(b) and 11(b)). For large flows ($>$ 10 K), the estimation of CSM is more accurate than that of PMC as shown in Figs. 10(c) and 11(c). However, RCC is better than PMC for small flows, and also better than CSM for large flows (See Figs. 10(b), 11(c) and 12(b)(c)). Figs. 11(d) and 10(d) show that standard errors of CSM and PMC are larger than RCC.

More interestingly, PMC and CSM with 16 Mb (Figs. 11 and 10) is outperformed by RCC with 8 Mb (Fig. 9).

In summary, PMC is better than CSM in small ranges, but CSM is better than PMC in large ranges. RCC surpassed CSM and PMC in terms of standard error and estimation accuracy over all flow sizes and over all memory sizes.

In Fig. 13, we show the relationship between the size of confinement and estimation accuracy. RCC uses a very small memory block (or virtual vector) size, ranging between 8 and 32 depending on the available memory. For example, with 8 Mb memory, RCC used 32-bit virtual vector, and with 16 Mb memory, eight-bit. Each bit of a virtual vector is randomly chosen from a memory block of confinement size c. With a larger confinement size, RCC's estimation is more accurate, and it is the best without confinement (or $c = m_A$) as shown in Fig. 13. However, RCC with a large c such as 512 is almost as accurate as RCC without confinement, which is shown in Fig. 13(b) and (c). Depending on the architecture of the network processor, the confinement size might vary. Considering that modern network processors have very wide busses [8], our evaluation with RCC having $c = 32$ is conservative, and thus, RCC's accuracy will be enhanced more with a larger confinement size.

Most of recent traffic measurement algorithms, such as CSM and PMC, have used the skewed distribution (or long-tail distribution), and RCC takes advantage of it more aggressively by adopting two structures. Thus, the performance of RCC might degrade more severely than others when the long-tail distribution disappears. For example, if the number of flows registered in the hash table exceeds a certain limit, encoding and decoding speed become slow, or the hash table might be overflown. In these cases, we can simply restart a new measurement period before they happen, which causes a shorter measurement period. Considering that current traffic has the long-tail distribution, RCC works well, but it will be a good research topic to devise a new data structure for the accumulator.

VII. Conclusion

We introduced a per-flow measurement algorithm called the recyclable counter with confinement. There are two ideas behind the design of the scheme: recycling a memory block by resetting for memory constraint, and confinement of virtual vectors to one word for speed constraint. Consequently, RCC outperforms previously proposed algorithms. Furthermore, there is the built-in data structure that holds flow labels and their sizes, so it is easy to retrieve the list of large flows directly from the table. The label retrieval together with the online decoding ability of our scheme will lead to per-flow measurements being used in new real-time services. Finally, we leave the investigation of a better data structure and algorithm for the hash table for future work.

References

[1] Cisco, "Cisco Visual Networking Index: Forecast and methodology, 2012–2017," Cisco whitepaper, 2013 [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360_ns827_Networking_Solutions_White_Paper.html

[2] Cisco, "NetFlow systems," [Online]. Available: http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html

[3] InMon Corp, "sFlow accuracy & billing," 2003 [Online]. Available: http://www.sflow.org/sFlowOverview.pdf

[4] The Cooperative Association for Internet Data Analysis, "Equinix Chicago data center," Nov. 21, 2013 [Online]. Available: http://www.caida.org

[5] Intel, "Intel IXP2800 and IXP2850 network processors datasheet," 2005.

[6] M. Adiletta, M. Rosenbluth, D. Bernstein, G. Wolrich, and H. Wilkinson, "The next generation of Intel IXP network processors," *Intel Technol. J.*, vol. 6, no. 3, pp. 6–18, 2002.

[7] V. Gopal and J. Guilfor, "A novel hashing method suitable for lookup functions," Intel, White Paper, 2012 [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/hash-method-performance-paper.pdf

[8] Netronome, "NFP-6xxx," 2012 [Online]. Available: http://www.netronome.com/product/nfp-6xxx/

[9] E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithms: Theory and Practice*. Upper Saddle River, NJ, USA: Prentice-Hall, 1977.

[10] D. Knuth, *The Art of Computer Programming*, 2nd ed. Reading, MA, USA: Addison-Wesley, 1998, vol. 3.

[11] P. Flajolet and G. Nigel Martin, "Probabilistic counting algorithms for database applications," *J. Comput. Syst. Sci.*, vol. 31, pp. 182–209, 1985.
[12] K. Y. Whang, B. Vander-Zanden, and H. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Trans. Database Syst.*, vol. 15, no. 2, pp. 208–229, 1990.
[13] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proc. ACM SIGCOMM*, Aug. 2002, pp. 323–336.
[14] S. Cohen and Y. Matias, "Spectral Bloom filters," in *Proc. ACM SIGMOD*, 2003, pp. 241–252.
[15] R. Karp, S. Shenker, and C. Papadimitriou, "A simple algorithm for finding frequent elements in streams and bags," *ACM Trans. Database Syst.*, vol. 28, no. 1, pp. 51–55, 2003.
[16] A. Kumar, J. JimXu, and J. Wang, "Space-code Bloom filter for efficient per-flow traffic measurement," in *Proc. IEEE INFOCOM*, Mar. 2004, pp. 1762–1773.
[17] 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.
[18] X. Dimitropoulos, P. Hurley, and A. Kind, "Probabilistic lossy counting: An efficient algorithm for finding heavy hitters," *Comput. Commun. Rev.*, vol. 38, no. 1, pp. 7–16, 2008.
[19] A. Fog, "Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs," Technical University of Denmark, 2014 [Online]. Available: http://www.agner.org/optimize
[20] 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.
[21] 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.
[22] P. Lieven and B. Scheuermann, "High-speed per-flow traffic measurement with probabilistic multiplicity counting," in *Proc. IEEE INFOCOM*, Apr. 2010, pp. 1–9.
[23] M. Yoon, T. Li, S. Chen, and J.-K. Peir, "Fit a compact spread estimator in small high-speed memory," *IEEE/ACM Trans. Netw.*, vol. 19, no. 5, pp. 1253–1264, Oct. 2011.
[24] 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.
[25] O. Rottenstreich, Y. Kanizo, and I. Keslassy, "The variable-increment counting Bloom filter," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 1880–1888.
[26] O. Rottenstreich and I. Keslassy, "The Bloom paradox: When not to use a Bloom filter?," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 1638–1646.

DaeHun Nyang received the B.Eng. degree in electronic engineering and computer science from Korea Advanced Institute of Science and Technology, Daejeon, Korea, in 1994, and the M.S. and Ph.D. degrees in computer science from Yonsei University, Seoul, Korea, in 1996 and 2000, respectively. He had been a Senior Researcher with the Electronics and Telecommunications Research Institute, Seoul, Korea, from 2000 to 2003. Since 2003, he has been a Full Professor with the Computer Information Engineering Department, Inha University, Incheon, Korea, where he is also the founding Director of the Information Security Research Laboratory. He is a member of the Board of Directors and Editorial Board of the Korea Institute of Information Security and Cryptology. Prof. Nyang's research interests include traffic measurement, cryptography, network security, privacy, usable security, biometrics, and their applications to authentication and public key cryptography.

DongOh Shin received the B.Eng. and M.S. degrees in computer engineering in 2010 and 2012, respectively, from Inha University, Incheon, Korea, where he is currently pursuing the Ph.D. degree. His research interests lie in network security, traffic measurement, and mobile security.