Cardinality Estimation for Elephant Flows: A Compact Solution Based on Virtual Register Sharing

Qingjun Xiao, *Member, IEEE, ACM*, Shigang Chen, *Fellow, IEEE*, You Zhou, *Member, IEEE*, Min Chen, *Member, IEEE*, Junzhou Luo, *Member, IEEE*, Tengli Li, and Yibei Ling, *Senior Member, IEEE*

***Abstract*— For many practical applications, it is a fundamental problem to estimate the flow cardinalities over big network data consisting of numerous flows (especially a large quantity of mouse flows mixed with a small number of elephant flows, whose cardinalities follow a power-law distribution). Traditionally the research on this problem focused on using a small amount of memory to estimate each flow's cardinality from a large range (up to 10^9). However, although the memory needed for each individual flow has been greatly compressed, when there is an extremely large number of flows, the overall memory demand can still be very high, exceeding the availability under some important scenarios, such as implementing online measurement modules in network processors using only on-chip cache memory. In this paper, instead of allocating a separated data structure (called *estimator*) for each flow, we take a different path by viewing all the flows together as a whole: Each flow is allocated with a virtual estimator, and these virtual estimators share a common memory space. We discover that sharing at the multi-bit register level is superior than sharing at the bit level. We propose a unified framework of virtual estimators that allows us to apply the idea of sharing to an array of cardinality estimation solutions, e.g., HyperLogLog and PCSA, achieving far better memory efficiency than the best existing work. Our experiment shows that the new solution can work in a tight memory space of less than 1 bit per flow or even one tenth of a bit per flow — a quest that has never been realized before.**

***Index Terms*— Big network data, flow monitoring, elephant flow, flow cardinality estimation.**

Manuscript received September 1, 2016; revised May 5, 2017; accepted September 6, 2017; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor M. Li. Date of publication October 9, 2017; date of current version December 15, 2017. The preliminary version of this paper titled "Hyper-Compact Virtual Estimators for Big Network Data Based on Register Sharing" was published in proceedings of the ACM SIGMETRICS, pp. 417–428, June 2015. This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFB1003000, in part by the National Natural Science Foundation of China under Grant 61502098, Grant 61632008, and Grant 61320106007, in part by Jiangsu Provincial Natural Science Foundation of China under Grant BK20150629, in part by the National Science Foundation of United States under Grant CNS-1719222 and Grant STC-1562485, in part by a Grant from the Florida Cybersecurity Center, in part by the Jiangsu Provincial Key Laboratory of Network and Information Security under Grant BM2003201, in part by the Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under Grant 93K-9, and in part by the Collaborative Innovation Center of Novel Software Technology and Industrialization. *(Corresponding author: Shigang Chen.)*

Q. Xiao, J. Luo, and T. Li are with the School of Computer Science and Engineering, Southeast University, Nanjing 210018, China (e-mail: csqjxiao@seu.edu.cn; jluo@seu.edu.cn; freya.li.tengli@gmail.com).

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

M. Chen was with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611 USA. He is now with Google Inc, Mountain View, CA 94043 USA (e-mail: minchen@google.com).

Y. Ling is with the Applied Research Laboratories, Telcordia Technologies, Morristown, NJ 07960 USA (e-mail: lingy@research.telcordia.com).

Digital Object Identifier 10.1109/TNET.2017.2753842

I. INTRODUCTION

CARDINALITY estimation is one of the fundamental problems in the area of network traffic measurement [1]–[6]. In a general definition, it is to estimate the number of *distinct* elements in each flow during a measurement period. The *flows* under measurement may be per-source flows, per-destination flows, per-source/destination flows, TCP flows, WWW flows, P2P flows, or abstract flows, such as client IPs accessing each URL object on a web server or client IPs querying each keyword. The *elements* may be destination addresses, source addresses, ports, values in other header fields, or even keywords that appear in the payload of packets in the flow.

Practical Importance: The cardinality problem has many practical applications. For example, if we treat all packets sent from the same *source address* as a flow (per-source flow), we may use a cardinality estimation module at a gateway or firewall to detect scanners by measuring the number of *distinct destination addresses* in each flow. In this case, packets belonging to a flow are identified by their common source address (also called *flow label*). The elements under measurement are the destination addresses in the headers of the packets. In the opposite example, we may treat all packets to a common destination as a flow and count the number of distinct source addresses in each flow. When we observe the cardinality of a certain flow suddenly surges, it may signal a DDoS attack against the destination address of the flow. For other applications, a large server farm may learn the popularity of its content by tracking the number of distinct users that access each file, where all accesses to a file form an (abstract) flow; an institutional gateway may determine the popularity of external web content for caching priority by tracking the number of outbound web requests for each web content, where all requests from different users to the same URL form a flow.

In another example, if Google treats all client IPs that query a keyword as a flow, the cardinality of the flow suggests the popularity of the keyword being searched. In this case, the flow label is the keyword under query. The estimator that works on per-keyword flows may be implemented as a function module at the web server. By a recent paper [6], many data analysis systems at Google, such as Sawzall, Dremel and PowerDrill, estimate the cardinalities of very large data sets on a daily basis. As pointed out in [6], cardinality estimation over large data sets presents a challenge in terms of computational resources, and memory in particular; for the PowerDrill system, a non-negligible fraction of queries historically could not be computed because they exceeded the available memory.

State of the Art: To deal with big data consisting of a very large number of flows, we must conserve memory space when designing a cardinality estimation module. For this purpose, a series of solutions were developed, including PCSA [7], MultiResBitmap [4] (a generalization of LinearCounting [8]), MinCount [9], LogLog [10], and HyperLogLog [11]. They all allocate a separate data structure, called *estimator*, for each individual flow. Every estimator contains a certain number of registers, bitmaps or other elementary structures. The most compact estimator in [11] requires hundreds of bytes to ensure a large estimation range and a good estimation accuracy.

Challenges: However, as the Internet moves into the era of big network data, hundreds of bytes per flow can be too much in some important scenarios — Modern high-speed routers forward packets at the speed of hundreds of Gigabits or even hundreds of Terabits per second [12]. The number of data flows that traverse a core router can be in tens of millions. Simultaneous tracking of such a large number of flows (each of which needs hundreds of bytes memory) brings a great challenge. The reason is that, in order to sustain high throughput, routers forward packets from incoming ports to outgoing ports via switching fabric, bypassing main memory and CPU. If one wants to apply cardinality estimation as an online module to process packets in real time, one way is to implement it on network processors at the incoming/outgoing ports and use on-chip cache memory. However, the commonly-used cache on processor chips is SRAM, typically a few megabytes, which may have to be shared among multiple functions for routing, performance, measurement, and/or security purposes. In such a context, the memory that can be allocated for the function of cardinality estimation may be even less than 1 bit per flow.

In another scenario, suppose a web search company wants to know how many different users have searched the same phrase (question or sentence) each day, which provides information on phrase popularity, useful in optimizing search performance or studying social trends on the Internet [13]. This is a cardinality estimation problem, where all search records for a given phrase form a flow. The number of flows (phrases, questions, sentences) can be in billions. Of course, we can resort to a data center for such big data, but it will be welcome if one can find a novel solution that deals with an extremely large number of flows in the memory of a cheap commodity computer.

Our Contribution: After decades of development [4], [7]–[11], it appears very difficult to further compress the size of an individual estimator much below hundreds of bits, without sacrificing estimation range or accuracy. Recently, an interesting idea was proposed to let different estimators (each for one flow) share bits [3], [5], [14], so that bits unused by one can be picked up by another. Along this line, we make three new contributions: First, we discover that sharing bits is actually inefficient because of too much noise introduced between estimators. Sharing space is good, but it should be done differently at the register level, not at the bit level, where a register is a multi-bit data structure that will be introduced later. Second, sharing has only been applied to bitmap and PCSA [7], an early work dated back to 1985. We develop a framework of virtual estimators which enables memory sharing for the recent cardinality estimation solutions, including PCSA [7], LogLog [10] and HyperLogLog [11], with the last one being the best existing work. Third, we fully develop the virtual HyperLogLog solution and the virtual PCSA solution, with a new procedure for recording per-flow information in the shared space, a set of formulas for estimating per-flow cardinality with noise removal, and the analytical results for estimation error under register sharing. We show that the new solutions can work in a tight memory space of less than 1 bit per flow or even one tenth of a bit per flow — a quest that has never been realized before.

The rest of the paper is organized as follows: Section II discusses the related work. Section III introduces our new design of register sharing. Section IV proposes a unified framework for constructing virtual estimators based on register sharing. Section V presents the detailed design of a memory-efficient cardinality estimation solution called virtual HyperLogLog under the framework. Section VI presents another memory-efficient solution named virtual PCSA under the same framework to demonstrate the framework's generalized applicability. For the two new solutions, Section VII analyzes their mean and variance. Section VIII evaluates the performance of the proposed estimation solutions through experiments based on real network traffic traces. Section IX draws the conclusion.

II. Related Work

Cardinality estimation is different from the related problem of *flow-size estimation* [15], which counts the number of elements (e.g., packets or bytes) in each flow through CountMin sketches (a generic tool for estimating the frequency of each element in a multiset) [16], CountSketch [17], Counter Braids [18], Lossy Counting [19] or Randomized Counter Sharing [15], with the goal of learning flow distribution or identifying heavy hitters. Consider all packets from a source address as a flow. Suppose the source sends 10,000 packets to a single destination address. The flow size is 10,000 when we measure the number of packets, but the flow cardinality is just one if we measure the *distinct* number of destination addresses in this flow. In short, cardinality estimation needs to remove duplicates, which makes it a more difficult problem since it has to somehow "remember" the

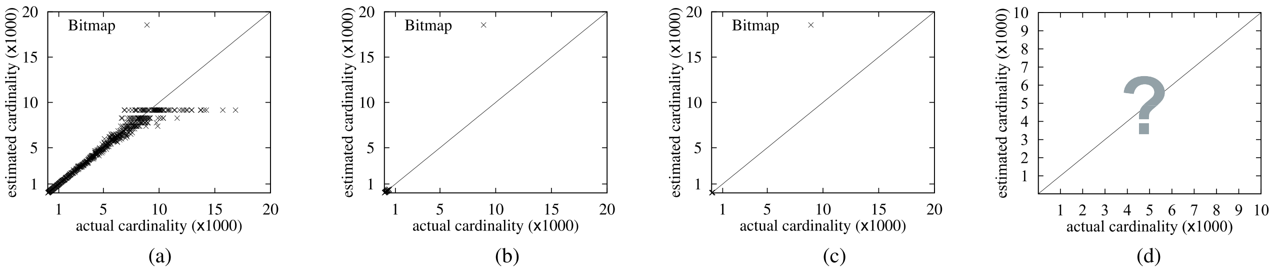


Fig. 1. Measurement results of the bitmap approach, whose estimation range is limited. Each flow is represented by one point. The x-coordinate is the true cardinality, and the y-coordinate is the estimated cardinality. The closer a point is to the equality line, the more accurate the estimation is. (a) 1280 bits per flow. (b) 96 bits per flow. (c) 32 bits per flow. (d) less than 1 bit per flow.

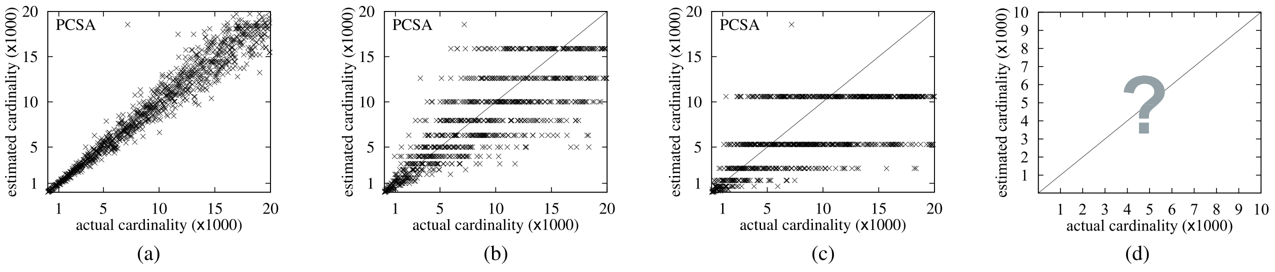


Fig. 2. Measurement results of PCSA or FM sketch named after Flajolet and Martin. (a) 1280 bits per flow, 40 registers of 32 bits each, 13% error. (b) 96 bits per flow, 3 registers of 32 bits each. (c) 32 bits per flow, 1 register of 32 bits. (d) less than 1 bit per flow.

observed elements for duplicate removal, while measuring a flow size only needs a counter.

Hash Table and Bitmap: It is too costly to design an estimator based on a hash table that stores all elements to remove duplicates. Instead, we may use a bitmap [8]: Initially all bits are zeros. Each arrival element is hashed to a bit which is then set to one. Duplicates are automatically filtered out since they are mapped to the same bit. At the end of a measurement period, the cardinality estimation is $\hat{n} = -b \ln V$ [8], where b is the number of bits used, V is the fraction of bits whose values remain zeros, and $\hat{n}$ is the estimated flow cardinality.

The problem of bitmap is that the estimation range is bounded by $b \ln b$. Hence, the bitmap has to be huge to handle a very large flow. Fig. 1 shows the simulation results, where the bitmap size is 1280 bits per flow in the leftmost plot, 96 bits per flow in the second plot, and 32 bits per flow in the third, respectively. Each flow is represented by a point, whose x-coordinate is the actual cardinality and y-coordinate is the estimated cardinality. The equality line is also shown. The closer a point is to the line, the more accurate the estimation is. The leftmost plot clearly shows a limited estimation range. As the bitmap size shrinks, the range shrinks quickly, as shown by plots (b)-(d). Note that "less than 1 bit" per flow will not work for the bitmap approach. Variants of the bitmap approach also own the problem of limited estimation range [2], [20], [21].

MultiResBitmap and PCSA: Sampling is one of the main methods in the literature for dealing with the estimation range problem. MultiResBitmap [4] is essentially the concatenation of multiple bitmaps, which have exponentially decreasing sampling probabilities. If we let the sampling probabilities be $\frac{1}{2}, \frac{1}{2^2}, \ldots, \frac{1}{2^w}$ and set each bitmap to its minimum size (a single bit), then we have the smallest MultiResBitmap, equivalent to an FM sketch of the earlier PCSA [7]. An FM sketch, also referred to as a *register* in the literature, can give an estimation up to 2^w, where w is the number of bits in the register. For example, $w = 32$ for an estimation range of 2^{32}.

To ease understanding, we illustrate one such PCSA register in Fig. 5. The register has an array of bits, and the probabilities for these bits to receive stream elements decreases exponentially by the series $\frac{1}{2}, \frac{1}{2^2}, \frac{1}{2^3}, \ldots$. As the input stream elements flow into a register, they will be pseudo-randomly mapped to the bit array. If a bit has received any elements, the bit will be set to one; Otherwise, the bit will remain zero. Note that the $\times$ mark in Fig. 5 represents that a bit is either zero or one. By maintaining the state of this bit array upon the arrivals of stream elements, PCSA algorithm always knows the position of leftmost zero bit, which is denoted by symbol M' in Fig. 5. Such a bit array is called a *PCSA register*, which can give an independent estimation of the stream cardinality as $2^{M'}$.

However, the estimation result from a single register is very inaccurate. To improve accuracy, FM uses multiple registers and returns the average of their estimations. Fig. 2 presents the simulation results of FM. It clearly has a larger estimation range, but its estimation accuracy is low even when there are 40 registers in the first plot. The estimation results are discrete when there are just a few registers in the second and third plots.

LogLog and HyperLogLog: The memory efficiency of PCSA still leaves much space for improvement: Its register size must be $\log_2 \; n_{\max} + O(1)$ bits, where $n_{\max}$ is the upper bound of measured cardinality, e.g., $n_{\max} = 2^{32}$. In contrast, follow-up algorithms named LogLog [10] and HyperLogLog [11] can reduce the memory cost per register to only $\log_2 \log_2 \; n_{\max} + O(1)$ bits [10]. Such significant memory

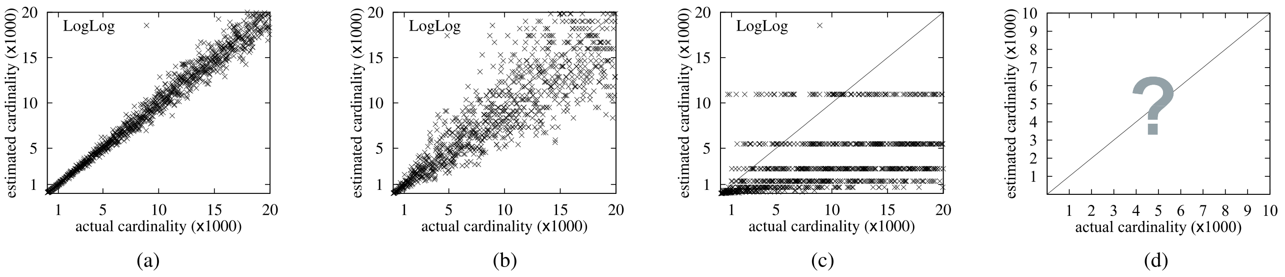


Fig. 3. Measurement results of LogLog. (a) 1280 bits per flow, 256 registers of 5 bits each, 8.1% error. (b) 80 bits per flow, 16 registers of 5 bits each, 33% error. (c) 5 bits per flow, 1 register of 5 bits. (d) less than 1 bit per flow.

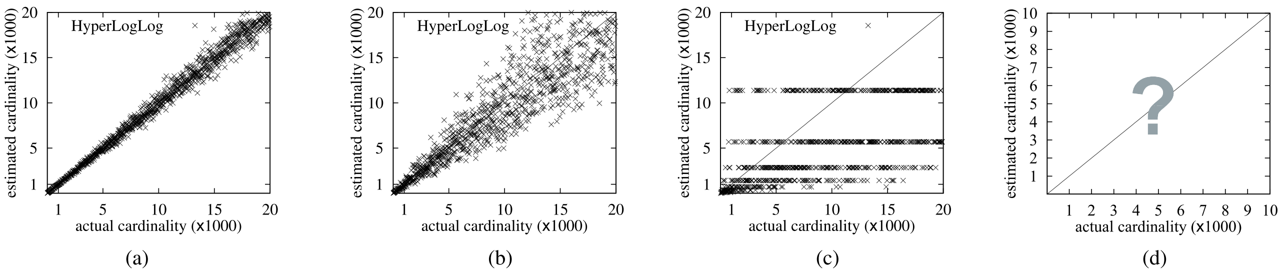


Fig. 4. Measurement results of HyperLogLog. (a) 1280 bits per flow, 256 registers of 5 bits each, 6.5% error. (b) 80 bits per flow, 16 registers of 5 bits each, 26% error. (c) 5 bits per flow, 1 register of 5 bits. (d) less than 1 bit per flow.

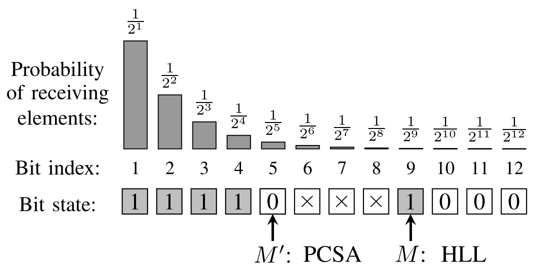

Probability of receiving elements:
Bit index: 1 2 3 4 5 6 7 8 9 10 11 12
Bit state: 1 1 1 1 0 × × × 1 0 0 0
M′: PCSA
M: HLL

Fig. 5. A register used by PCSA vs. a register of LogLog and HyperLogLog.

TABLE I
COMPARISON OF THE PRIOR ART

Solution	Std. Err. (σ)	Mem Units	Mem (σ=5%)
MinCount	$1.00/\sqrt{m}$	≤32 bits	1600 bytes †
MultiResBitmap	$\approx 4.4/\sqrt{m}$	1 bit	968 bytes
PCSA	$0.78/\sqrt{m}$	32-bit registers	974 bytes
LogLog	$1.30/\sqrt{m}$	5-bit registers	423 bytes
HyperLogLog	$1.04/\sqrt{m}$	5-bit registers	271 bytes

† For MinCount, we assume the size of its memory units is 32 bits, and each unit stores the 32-bit hash value of a stream element.

compression is because, instead of maintaining the state of an entire bit array as in Fig. 5, LogLog and HyperLogLog[1] use a multi-bit register to record only the position of rightmost one bit, which is called "LogLog register". We denote it by the symbol M in Fig. 5, which can give 2^M as an independent estimation of the stream cardinality.

Therefore, both LogLog [10] and HyperLogLog [11] can compress the size of each register from 32 bits to only 5 bits for the same estimation range of 2^{32}. Their performance is presented in Figures 3 and 4. The estimation accuracy of LogLog and HyperLogLog (HLL) is much better than PCSA, because smaller registers mean more of them can be allocated from the same memory budget, which drives the estimation variance down. However, they still do not work well for 80 bits in the second plot of Fig. 3 and Fig. 4 (with the relative standard error being 33% for LogLog and 26% for HLL), let alone less than one bit per flow. A work that analyzes the theoretical bound of the memory efficiency of cardinality estimation problem can be found in [22].

Performance Summary: The performance of the traditional cardinality estimators is summarized in Table I, where MinCount [9], [23] takes a different approach by hashing each arrival element and keeping a number of smallest hash values, from which the estimation is made (using the range of the smallest hash values). In the second column, m is the number of smallest hash values kept by MinCount, the number of bits used by MultiResBitmap, or the number of registers used by other approaches. The total memory cost is m multiplied by the size of each memory unit (hash value, bit or register).

For a single flow, the memory needed to control the standard error within 5% of the actual cardinality is given in the last column, which shows the progress in memory saving over the past decades: If we use PCSA as the initial benchmark, the seminal work of LogLog cuts the memory requirement by more than half. The followup HyperLogLog cuts the memory further by more than 30%. HyperLogLog has made great impact on IT industry and was adopted by Google [6], PostgreSQL, P2P systems [24], and DDoS attack detection systems [11].

[1]HyperLogLog [11] is a variant of LogLog [10] to further improve accuracy. Although they both use the observation of position M shown in Fig. 5, they adopt different methods to aggregate the estimation results of a set of registers. LogLog uses geometric averaging, while HyperLogLog uses harmonic mean, in order to mitigate the impact of *outlier registers* with abnormally large estimations, thereby appreciably increasing the quality of estimations.

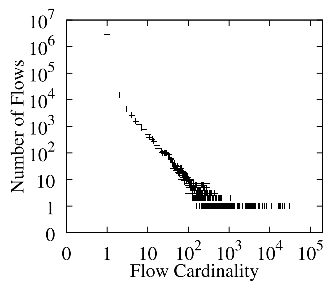


Fig. 6. Flow distribution: each point shows the number (y-coordinate) of flows having a certain cardinality (x-coordinate).

III. Our New Approach of Register Sharing and Virtual Estimators

A. Motivation: Waste of Space

The traditional solutions allocate one estimator for each flow, which however causes a serious waste of space. As an example, we download network traffic traces from CAIDA [25]. Consider per-source flows. The cardinality of each flow is the number of distinct destination IP addresses contacted by a source. We illustrate the distribution of the flow cardinalities in Fig. 6, where the measurement period is 10 minutes and each point shows the number (y-coordinate) of flows that have a certain cardinality (x-coordinate). A roughly straight line on a log-log plot is often considered as the signature of a power law distribution (rougly $y = 3 \cdot 10^4 \cdot x^{-1.7}$). This log-scale figure shows that the vast majority of flows have small cardinalities, while a small number of flows have large cardinalities.

Without knowing the flows' cardinalities beforehand (which are in fact what we want to figure out), the estimators of all flows are set according to the maximum range of cardinality, requiring hundreds of bits even for the best estimator. However, if a flow turns out to be small, e.g., with a cardinality of 1, most of the bits will be left unused and thus wasted.

B. Memory Sharing among Virtual Estimators

One way to exploit unused bits is to share bits among the estimators. Two solutions were proposed for sharing among bitmaps [3] and FM sketches [5]. In the compact spread estimator (CSE) [3], a bitmap is allocated for each flow, but all bitmaps share their bits from a common bit pool. However, it is difficult to extend the estimation range of bitmaps without incurring large overhead or causing estimation inaccuracy.

In the probabilistic multiplicity counting solution (PMC) [5], an estimator with multiple FM sketches is allocated to each flow. In fact, PMC was originally designed for estimating the flow size (i.e., the number of packets in each flow), but it can be easily modified for estimating the flow cardinality, which is not commonly true for other flow-size estimators. As illustrated in Fig. 7, the FM sketches (called registers) of all estimators share their *bits* from a common bit pool uniformly at random, so that mostly unused higher-order bits in the registers can be utilized.

Our idea is to propose yet another memory-sharing method, i.e., share memory at the register level, as depicted in Fig. 8.

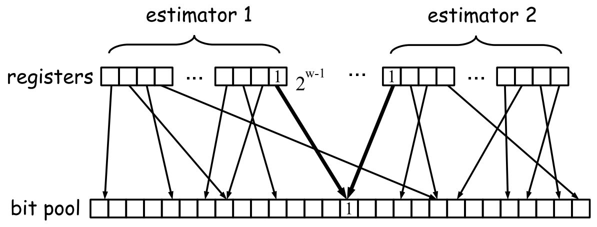


Fig. 7. Bit sharing as in [5], where the FM sketches (registers) share their individual bits from a common bit pool.

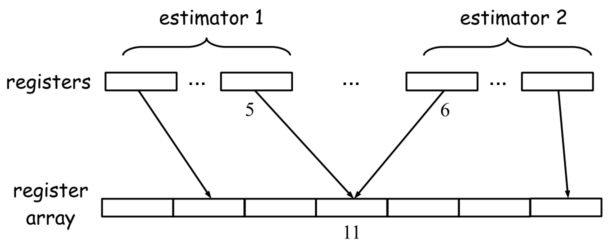


Fig. 8. Register sharing, where the estimators share their registers from a common register pool.

The estimators of different flows share their registers from a common register array. Here, we define a register as a multi-bit data structure, which could be either a PCSA register or a HyperLogLog register as shown in Fig. 5. Unlike a single bit, a register having multiple bits is able to give an independent estimation about the cardinality of stream elements that are mapped to this register. Given a fixed array of registers, we dynamically create an estimator for a new flow by randomly drawing a number of registers from the array. In a sense, the array of registers are physical, but the estimators are logical because they are created on the fly without additional memory allocation. Hence, they are called *virtual estimators*.

Suppose a system allocates a certain amount of physical memory to the function of cardinality estimation. The number of bits available may be smaller than the number of flows. If this is the case, the number of registers in M will certainly be even smaller. Each register is thus shared by many virtual estimators, ensuring that the register is fully utilized.

Consider the virtual estimator of an arbitrary flow. What it estimates is actually the cardinality of the flow plus the noise introduced by other flows that share its registers. Refer to Figure 8 where estimator 1 and estimator 2 share a common register. If the register records 5 elements from the flow of estimator 1 and 6 elements from the flow of estimator 2, the final result will be 11 elements recorded. From the viewpoint of estimator 1, the register carries its flow's information as well as noise from other flows. The same is true from the viewpoint of estimator 2.

Because the registers in all virtual estimators are randomly picked, there is an equal opportunity for any two registers from different estimators to be mapped to the same physical register in M. Hence, as one virtual estimator records an element of its flow into one of its registers, the probability for this operation to cause noise to any other virtual estimator is the same. When there are a large number of virtual estimators and each of them randomly chooses a large number of registers, the noise that they cause to each other will be roughly uniform. Such uniform noise can be measured and removed.

Of course, there may exist registers that are hit by the virtual estimators of elephant flows. However, the number of large elephant flows is often exponentially fewer than the number of small flows; see Fig. 6 for example. That means the number of registers that carry abnormally large noise account for a small fraction of all registers in M. If the estimator of a small flow contains one or a few registers of large noise, the technique of harmonic averaging can be used to remove the effect of such outliers (which is already done by [10], [11]).

IV. A Framework for Virtual-Estimator Solutions

We propose a framework for developing virtual-estimator solutions that enable register-level sharing for mainstream sketches, e.g., PCSA [7], LogLog [10], and HLL [11]. The next section will show as an example how to apply the framework to HLL for a virtual-estimator solution named vHLL. The next next section will show another example of applying the framework to PCSA for a second solution named vPCSA.

In the framework, we use a single array M of m registers to store the cardinality information of all flows. The ith register in the array is denoted by $M[i]$, $0 \leq i < m$. The size of the registers is set based on the type of estimators used [7], [10], [11] and the maximum range of cardinality to be estimated. For example, in the vHLL solution, the size of registers is five bits, in order to measure big cardinalities up to $2^{2^5} \approx 4 \times 10^9$. Each flow has s virtual registers that are randomly selected from M through hash functions. These registers logically form a virtual estimator, denoted as M_f, where f is the label of the flow. The ith register of the virtual estimator, denoted as $M_f[i]$, $0 \leq i < s$, is selected from M as follows:

$$M_f[i] = M[H_i(f)], \tag{1}$$

where $H_i(\ldots)$ is a hash function which is in the range $[0, m)$. We want to stress that M_f is not a separate data structure. It is merely a logical construction based on registers selected from M, and it is not explicitly constructed during online operation. In all our later formulas, one should treat the notation $M_f[i]$ simply as $M[H_i(f)]$, referring to a register in M.

The hash function H_i, $0 \leq i < s$, can be implemented from a master function $H(\ldots)$:

$$H_i(f) = H(f \mid i) \quad \text{or} \quad H_i(f) = H(f \oplus R[i]), \tag{2}$$

where '|' is the concatenation operator, '$\oplus$' is the XOR operator, and $R[i]$ is a constant whose bits differ randomly for different indexes i. The master hash function H we have adopted in experiments is METRO hash or MURMUR3 hash.

At the beginning of each measurement period, all registers are reset to zeros. The arrival stream of elements is abstracted as a sequence of $\langle f, e \rangle$ pairs, where f is a flow label and e is an element of the flow. For example, if a router measures per-source flows for their numbers of distinct destination addresses, it extracts from each arrival packet the source address as the flow label and the destination address in the IP header as the element to be recorded. For each pair $\langle f, e \rangle$, we record e in one of the registers of M_f based on the cardinality estimation methods in [7], [10], or [11], depending on which one is used.

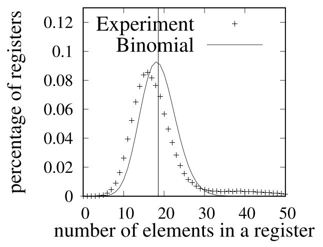


Fig. 9. Distribution of number of distinct elements in a register.

At the end of a measurement period, the register array M is offloaded to a server for long-term storage. Given a flow label f in offline query, we reconstruct its virtual estimator M_f by copying s registers from M at indices $H_i(f)$, $0 \leq i < s$. Let n_s be the number of distinct elements recorded by M_f, which is the flow's cardinality plus the noise introduced by other flows due to register sharing. Let n_f be the actual cardinality of flow f. The noise term is $n_s - n_f$. We use the estimation formula from [7], [10], or [11] (depending on which one is used) to give an estimation $\hat{n_s}$ of n_s. Below we focus on noise estimation.

Let n be the sum of all flows' cardinalities. From the flow f's point of view, the elements of all other flows, $n - n_f$ of them, are noise. Let Y be a random variable for the number of noise elements recorded by an arbitrary register in M. When the number of flows and the number of registers per estimator are both sufficiently large and the cardinality of any flow is negligibly small when comparing with n, Y approximately follows the binomial distribution $Bino(n - n_f, \frac{1}{m})$, because each noise element has approximately an equal chance to be recorded by any register due to the random selection of registers by virtual estimators. Such an approximation is supported by Fig. 9 about the distribution results from the experiments in Section VIII. The figure shows that the experimental distribution approximately follows the computed binomial curve. They do not match exactly, but the approximation is good enough to provide accuracy in cardinality estimation. In particular, the mean value (used in the estimator) of the experimental distribution aligns very well with that of the binomial distribution.

Hence, we have

$$E(Y) = \frac{n - n_f}{m}.$$

The total noise, $n_s - n_f$, is the sum of individual noises in the s registers of M_f. Hence, $n_s - n_f$ can be considered as the sum of s independent random variables of $Bino(n - n_f, \frac{1}{m})$.

$$E(n_s - n_f) = s\,E(Y) = s\,\frac{n - n_f}{m} \tag{3}$$

By the law of large numbers in the probability theory, the relative variance $Var(\frac{n_s - n_f}{E(n_s - n_f)})$ approaches to zero when s is large. In this case, $E(n_s - n_f)$ can be approximated by an instance value, $n_s - n_f$. We have

$$n_s - n_f \approx \frac{n - n_f}{m} s, \quad n_f \approx \frac{ms}{m-s}\Big(\frac{n_s}{s} - \frac{n}{m}\Big). \tag{4}$$

We define a *grand flow* as the combination of all flows. With a few hundreds of extra bytes and applying the HyperLogLog, we can obtain an accurate estimation $\hat{n}$ for n (see Table I), while the additional memory overhead is negligible when comparing with the memory space M. Alternatively, since the elements of the grand flow distribute approximately in uniform over M, we can use the entire register array M as an estimator to give an estimation for n (using HyperLogLog, for example).

Let $\hat{n_f}$ be our estimation of n_f. We have the following estimation formula from (4).

$$\hat{n_f} = \frac{ms}{m-s} \cdot \Big(\frac{\hat{n_s}}{s} - \frac{\hat{n}}{m}\Big) \tag{5}$$

In the virtual estimator of flow f, we regard the stream elements belonging to flow f as signals, and regard the elements from other flows (that are by chance mapped to flow fs registers) as noises. Then, the above equation removes the noise by subtracting the expected value of the cardinality of noises (i.e., $\frac{s}{m}\hat{n}$). We can only remove the noise mean but cannot completely remove the variance in noise, which is higher when the noise level is higher. Thus, the larger the signal-to-noise ratio is, the better accuracy we will have for the noise removal operation. Elephant flows with large signal-to-noise ratio can remove noises with not much accuracy loss. But mouse flows with very small signal-to-noise ratio will inevitably suffer from low estimation accuracy. In the next section, we will select vHLL, i.e., virtual HyperLogLog, to discuss its operations and performance in details.

Also note that our virtual estimator for per-flow cardinality estimation can be easily extended to solving the problem of identifying top-k largest flows. Suppose the line card of a high-speed router maintains two data structures simultaneously: a vHLL sketch as described above, and a min-heap structure to record the IDs of top-k flows. When a new packet arrives at the line card, its packet processing module will read the vHLL sketch and check whether the flow of the incoming packet has its estimated cardinality larger than the flow at the tree root of min-heap. If it is true, then this new flow will be inserted into the heap structure as a new member of top-k flows, and the flows that are no longer qualified to be top-k will be removed.

V. Virtual HyperLogLog Estimator

In this section, as an example, we apply the framework of virtual estimators on HyperLogLog for a new solution, vHLL, based on register-level sharing. This solution consists of two components: one for recording the stream of packets in the virtual HyperLogLog estimators, and the other for estimating the cardinality of an arbitrary flow f.

A. Record Flow Elements in Virtual Estimator

Consider a flow f. When a measurement period begins, all registers in its virtual estimator M_f are reset to zeros. For each arrival element e of flow f, we perform the hashing below:

$$\begin{aligned} H(e) &= \langle x_1 x_2 \ldots \rangle \\ p &= \langle x_1 x_2 \ldots x_b \rangle \\ q &= \langle x_{b+1} x_{b+2}, \ldots \rangle, \end{aligned} \tag{6}$$

where $\langle x_1 \ x_2 \ldots \rangle$ is binary format of the hash output $H(e)$, p denotes the leading b bits with b equal to $\log_2 \ s$, and q represents the remaining bits. Using the value of p, we can map e pseudo-randomly to a register $M_f[p \bmod s]$. For clarity, we will breviate $M_f[p \bmod s]$ simply as $M_f[p]$ afterwards.

The operation of recording e is simple: Let $\rho(q)$ be the number of leading zeros in q plus one; for example, if $q = 001\ldots$, then $\rho(q) = 3$. Clearly, the probability of $\rho(q) = i$ is $(\frac{1}{2})^i$, for $\forall i > 0$. We update $M_f[p]$ if its current value is smaller than $\rho(q)$. Namely,

$$M_f[p] := \max\big(M_f[p], \ \rho(q)\big), \tag{7}$$

where $:=$ is assignment operator. Hence, $M_f[p]$ has recorded (one plus) the longest run of leading zeros from any element mapped to the register. Suppose $M_f[p] = M[H_p(f)]$ as in (1), and $H_p(f) = H(f \mid p)$ as in (2). Combining (7), (1) and (2),

$$M[H(f \mid p)] := \max\big(M[H(f \mid p)], \ \rho(q)\big). \tag{8}$$

Eq. (8) shows that the operations are actually performed on the physical register array M, and the virtual estimator is just logical in the online recording phase.

Per-Packet Processing Cost: Whenever a packet e arrives, our vHLL algorithm needs to use equation (8) to update a particular register. In order to locate the register, it requires two hash operations: $H(f\,|\,p)$, and $H(e)$ for getting p and q values in (6). After locating the register $M[H(f|p)]$, it also requires at most two memory accesses for updating its value, i.e., reading $M[H(f \mid p)]$ and writing $M[H(f \mid p)]$ back if its value changes. Note that the writing operation happens rarely since the likelihood for $\rho(q) > M[H(f\,|\,p)]$ to happen will decrease exponentially as the register's value increases. Consider an arbitrary register of value v at the end of a measurement period. There are at most v writes to this register. Each distinct element recorded by the register has a probability of $(1/2)^v$ to set its value to v. Hence, the expected number of distinct elements recorded by the register is about 2^v. Hence, the write/read ratio is $v/2^v$. As an example, the average register value in our experiments (Section VIII) is about 7, which corresponds to a write/read ratio of $7/2^7 \approx 5.5\%$, suggesting infrequent write operations. Nonetheless, in a pipelined design, as most packets only require read and some packets require an additional write, the pipeline will be stalled (for the time of one memory access) each time write occurs.

B. Flow Cardinality Estimation

Given a flow label f for offline query, we construct M_f from the stored M. Consider an arbitrary register $M_f[i]$, $0 \leq i \ < s$. Any element mapped to this register had a probability of $\frac{1}{2^{M_f[i]}}$ to set the register to its current value. Hence, the estimation for the number of elements mapped to this register is $2^{M_f[i]}$ [11].

Recall that n_s is the total number of distinct elements that have been recorded by the estimator M_f, including both elements in flow f and those in other flows that share registers in M_f. In order to estimate n_s, the normalized harmonic mean is applied to aggregate the estimations from all registers in M_f:

$$\hat{n_s} = \alpha_s \cdot s^2 \cdot \left(\sum_{j=0}^{s-1} 2^{-M_f[j]}\right)^{-1}, \tag{9}$$

where α_s is a bias correction constant. The mathematical expression of α_s is quite complicated. So instead its numerical values are often used in practice: $\alpha_{16} = 0.673$, $\alpha_{32} = 0.697$, $\alpha_{64} = 0.709$, and $\alpha_s = 0.7213/(1 + 1.079/s)$ when $s \geq 128$.

The estimator in (9) is good for large cardinalities, but it is severely biased when dealing with small cardinalities [11]. For a small cardinality, we treat M_f as a bitmap of s bits, with each register $M_f[i]$ converted to one bit, whose value is 1 when $M_f[i] > 0$ or zero otherwise. The estimation formula is $\hat{n_s} = -s \log V$, where V is the fraction of bits in the bitmap that are zeros [8]. This formula is used when the cardinality estimation by (9) is smaller than 2.5 s.

Recall that we can estimate the sum $\hat{n}$ of all flow cardinalities based on a separate estimator or simply from the whole array M using (9) where $\hat{n_s}$ is replaced with $\hat{n}$, s is replaced with m, and M_f is replaced with M. After computing both $\hat{n_s}$ and $\hat{n}$, we use (5) to compute the estimated flow cardinality $\hat{n_f}$.

VI. Virtual PCSA Estimator

In this section, as another example, we apply the framework of virtual estimators to PCSA [7] for another solution named virtual PCSA (vPCSA). This solution consists of two components: an online component for recording the stream of packets into the virtual PCSA estimators, and the offline component for estimating the cardinality of an arbitrary flow f.

A. Online Component for Recording Flow Elements

Consider a flow f. For each arrival element e of flow f, we perform the same hash operation as in (6). Then, using the value of p, we can map the element e to a register $M_f[p \bmod s]$, which is abbreviated as $M_f[p]$ for clarity. Using the value of $\rho(q)$, we can record the element e at an appropriate bit of the register $M_f[p]$, using the equation below:

$$M_f[p] := M_f[p] \vee \left(1 \ll (\rho(q) - 1)\right), \tag{10}$$

where $\ll$ is the bitwise left shift operator, and $\vee$ is the bitwise OR operator. So the above equation assigns the $[\rho(q) - 1]$th bit of the register $M_f[p]$ to one. Typically, when implementing the PCSA algorithm, each register $M_f[p]$ is given 32 bits memory. The probabilities for these bits to be assigned to one reduces exponentially as $2^{-1}, 2^{-2}, 2^{-3}, \ldots$, from low end to high end.

Suppose the pth register of virtual estimator M_f is mapped to the $H_p(f)$th register of physical estimator M, i.e., $M_f[p] = M[H_p(f)]$ as in (1). Suppose the hash function H_p is implemented by the concatenation operator $|$, which is $H_p(f) = H(f \mid p)$ as in (2). By combining (10), (1) and (2), we have

$$M[H(f \mid p)] := M[H(f \mid p)] \vee \left(1 \ll (\rho(q) - 1)\right). \tag{11}$$

Per-Packet Processing Cost: When a packet e arrives, our vPCSA algorithm will use equation (11) to update a particular register. In order to locate the register, it requires two hash operations: $H(f \mid p)$, and $H(e)$ for getting p and q values in (6). After locating the register $M[H(f \mid p)]$, it requires only one memory access for updating its value, i.e., setting the $(\rho(q) - 1)$th bit of register $M[H(f \mid p)]$ to one. We observe a space-processing tradeoff between vHLL and vPCSA. As our experiment will show, vHLL is more efficient in space; alternatively, it estimates flow cardinalities more accurately under the same space availability. But vPCSA incurs smaller per-packet processing overhead. It requires one memory access on a single bit per packet, whereas vHLL requires possibly two memory accesses on a five-bit register per packet.

B. Offline Component for Estimating Flow Cardinality

Given a flow label f for offline query, we construct M_f from the stored M. Recall that n_s is the total number of distinct elements that have been recorded by the estimator M_f, including both elements in flow f and those in other flows that share registers in M_f. We estimate n_s by the equation:

$$\hat{n_s} = 1/\phi \times s \times 2^{\frac{1}{s}\sum_{j=0}^{s-1} \mathrm{LZB}(M_f[j])}, \tag{12}$$

where ϕ is the bias correction factor that is equal to 0.77351, and $\mathrm{LZB}(x)$ returns the position of the lowest zero bit, among all the thirty-two bits of integer x. Note that the lowest bit position starts from 0, and, hence, $\mathrm{LZB}(\ldots 010\underline{0}11111) = 5$.

However, the estimation equation in (12) has a bias problem when dealing with small cardinalities whose load factor n_s/s is smaller than 20. Hence, a small bias correction term is proposed by literature [26], which is presented as follows:

$$\hat{n_s} = 1/\phi \times s \times \left(2^{Z/s} - 2^{-\kappa \cdot Z/s}\right), \tag{13}$$

where Z represents the summation term $\sum_{j=0}^{s-1} \mathrm{LZB}(M_f[j])$, and, for the two constants, $\kappa = 1.75$, and $\phi = 0.77351$.

Similar to HyperLogLog, when the cardinality estimated by (13) is smaller than $2.5s$, another estimation equation based on LinearCounting [8] is used, i.e., $\hat{n_s} = -s \log V$, where V is the fraction of registers in M_f equal to zeros.

We can estimate the grand flow cardinality n from the whole register array M, using the equation (13), where $\hat{n_s}$ is replaced with $\hat{n}$, s is replaced with m, and M_f is replaced with M. After computing both $\hat{n_s}$ and $\hat{n}$, we further use the equation (5) to compute the estimated flow cardinality $\hat{n_f}$.

VII. Estimation Bias and Variance

This section analyzes the bias and standard error of our vHLL and vPCSA estimators. By [11], we have the following theorem about the bias and standard error of HyperLogLog.

Theorem 1: Let n_s be the number of distinct elements that are mapped to a HyperLogLog estimator M_f. Suppose the number s of registers in M_f is more than 16.

- *If n_s is sufficiently large, the estimate $\hat{n_s}$ by (9) is asymptotically almost unbiased in the sense that*

$$\frac{1}{n_s} E(\hat{n_s}) = 1 + \delta_1(n_s) + o(1),$$

where $|\delta_1(n_s)| < 5 \times 10^{-5}$ as soon as $s \geq 16$.

- *The standard error defined as* $\frac{1}{n_s}\sqrt{Var(\hat{n_s})}$ *satisfies*
$$\frac{1}{n_s}\sqrt{Var(\hat{n_s})} = \frac{\beta_s}{\sqrt{s}} + \delta_2(n_s) + o(1),$$
where $|\delta_2(n_s)| < 5 \times 10^{-4}$ *as soon as* $s \geq 16$. *The constants* β_s *being bounded, with* $\beta_{16} = 1.106$, $\beta_{32} = 1.070$, $\beta_{64} = 1.054$, $\beta_{128} = 1.046$, *and* $\beta_\infty = 1.039$.

As stated in the HyperLogLog paper [11], the functions δ_1 *and* δ_2 *represent oscillating functions of a tiny amplitude, and they can be safely neglected for all practical purposes.*

Additionally, according to [7], we have the following theorem that states the bias and standard error of PCSA.

Theorem 2: Let n_s *be the number of distinct elements that flow into a PCSA estimator, whose number of registers is* s. *The cardinality estimation* $\hat{n_s}$ *generated by PCSA in (12) has the following characterizations of the bias and standard error:*
$$\frac{1}{n_s}E(\hat{n_s}) \approx 1 + \epsilon(s), \quad \frac{1}{n_s}\sqrt{Var(\hat{n_s})} \approx \eta(s),$$
where quantities $\epsilon(s)$ *and* $\eta(s)$ *satisfy as* s *gets large:*
$$\epsilon(s) \sim \lambda/(2s), \quad \eta(s) \sim \sqrt{\lambda}/\sqrt{s},$$
where λ *is a constant, which can be closely approximated by 0.61 for all values of* s. *The detailed formula of* λ *is*
$$\lambda = \frac{\pi^2}{12} - \frac{\gamma^2}{2} - N'(0) - N''(0) + \frac{\log^2 2}{12},$$
where γ *is the Euler-Mascheroni constant with* $\gamma = 0.5772\ldots$. *The function* $N(x)$ *is defined as* $N(x) = \sum_{j\geq 1}\frac{(-1)^{v(j)}}{j^x}$, *where* $v(j)$ *is the number of ones in the binary representation of* j.

A. Estimation Bias

Given an arbitrary flow f, we know from Section IV that n_s is the sum of the flow cardinality n_f and a noise random variable $n_s - n_f$ with a binomial distribution of $Bino(n - n_f, \frac{s}{m})$. For $\forall i \in [0, n - n_f]$, we have
$$Prob\{n_s - n_f = i\} = \binom{n - n_f}{i}(\frac{s}{m})^i(1 - \frac{s}{m})^{n-n_f-i}. \quad (14)$$
Under the condition of $n_s - n_f = i$, by Theorem 1, for the vHLL solution, we have
$$\begin{aligned} E(\hat{n_s} \mid n_s - n_f = i) &= (n_f + i)\big(1 + \delta_1(n_f + i) + o(1)\big) \\ &\approx n_f + i, \end{aligned} \quad (15)$$
with a small error bounded by a ratio of 5×10^{-5}. Similarly, for the vPCSA, according to Theorem 2, we have
$$\begin{aligned} E(\hat{n_s} \mid n_s - n_f = i) &= (n_f + i)\big(1 + \epsilon(s)\big) \\ &\approx (n_f + i)\big(1 + 0.31/s\big) \approx n_f + i. \end{aligned} \quad (16)$$
Hence, for both virtual estimators vHLL and vPCSA,
$$\begin{aligned} E(\hat{n_s}) &= \sum_{i=0}^{n-n_f} E(\hat{n_s} \mid n_s - n_f = i) \times Prob\{n_s - n_f = i\} \\ &\approx \sum_{i=0}^{n-n_f} (n_f + i) \times \binom{n - n_f}{i}(\frac{s}{m})^i(1 - \frac{s}{m})^{n-n_f-i} \\ &= n_f + (n - n_f)\frac{s}{m}. \end{aligned} \quad (17)$$

The value of $\hat{n}$ is estimated based on the entire array M or through a separate estimator with hundreds of bytes (i.e., much more than 16 registers). For the vHLL solution, from Theorem 1, we have $E(\hat{n}) = n\ (1 + \delta_1(n) + o(1)) \approx n$, with a very small error bounded by a ratio of 5×10^{-5}. For the vPCSA solution, from Theorem 2, we have $E(\hat{n}) \approx n\ (1 + 0.31/m) \approx n$.

Applying $E(\hat{n}) \approx n$ and $E(\hat{n_s}) \approx n_f + (n - n_f)\frac{s}{m}$ to the estimation formula (5), we have
$$\begin{aligned} E(\hat{n_f}) &= \frac{ms}{m-s}\Big(\frac{E(\hat{n_s})}{s} - \frac{E(\hat{n})}{m}\Big) \\ &\approx \frac{ms}{m-s}\Big(\frac{n_f + (n - n_f)\frac{s}{m}}{s} - \frac{n}{m}\Big) = n_f. \end{aligned} \quad (18)$$
Hence, both vHLL and vPCSA are asymptotically unbiased.

B. Estimation Variance

Next we derive the variance of $\hat{n_f}$.
$$\begin{aligned} Var(\hat{n_f}) &= (\frac{ms}{m-s})^2\Big(\frac{Var(\hat{n_s})}{s^2} + \frac{Var(\hat{n})}{m^2}\Big) \\ &= (\frac{ms}{m-s})^2\Big(\frac{E(\hat{n_s}^2) - \big(E(\hat{n_s})\big)^2}{s^2} + \frac{Var(\hat{n})}{m^2}\Big) \\ &= (\frac{m}{m-s})^2\Big(E(\hat{n_s}^2) - \big(E(\hat{n_s})\big)^2 + (\frac{s}{m})^2 Var(\hat{n})\Big) \end{aligned} \quad (19)$$
With $\forall i \in [0, n - n_f)$, under the condition of $n_s - n_f = i$, by Theorem 1, for the vHLL solution, we have
$$\begin{aligned} \frac{1}{n_f + i}\sqrt{Var(\hat{n_s} \mid n_s - n_f = i)} &= \frac{\beta_s}{\sqrt{s}} + \delta_2(n_f + i) + o(1) \\ &= \frac{\beta_s}{\sqrt{s}} \approx \frac{1.04}{\sqrt{s}}, \end{aligned} \quad (20)$$
where we use 1.04 to approximate β_s, assuming $s \geq 128$, which is always the case in our experiments later. Hence,
$$Var(\hat{n_s} \mid n_s - n_f = i) \approx \frac{1.04^2}{s}(n_f + i)^2. \quad (21)$$
Similarly, for the vPCSA solution, by Theorem 2, we have
$$\frac{1}{n_f + i}\sqrt{Var(\hat{n_s} \mid n_s - n_f = i)} \approx \frac{\sqrt{\lambda}}{\sqrt{s}} \approx \frac{\sqrt{0.61}}{\sqrt{s}} \approx \frac{0.78}{\sqrt{s}}. \quad (22)$$
Hence, for vPCSA,
$$Var(\hat{n_s} \mid n_s - n_f = i) \approx \frac{0.78^2}{s}(n_f + i)^2. \quad (23)$$
When estimating the grand flow cardinality n, we have
$$\begin{aligned} Var(\hat{n}) &\approx \frac{1.04^2}{m}n^2 \text{ for vHLL}, \\ Var(\hat{n}) &\approx \frac{0.78^2}{m}n^2 \text{ for vPCSA}, \end{aligned} \quad (24)$$
where m is the number of registers in the physical estimator M, and we let $m \geq 128$. Because $E(\hat{n_s}^2 \mid n_s = n_f + i) = Var(\hat{n_s} \mid n_s - n_f = i) + \big(E(\hat{n_s} \mid n_s - n_f = i)\big)^2$, from (15) and (21), when s is sufficiently large, we have
$$\begin{aligned} E(\hat{n_s}^2 \mid n_s - n_f = i) &\approx \frac{\beta^2(n_f + i)^2}{s} + (n_f + i)^2 \\ &= (\frac{\beta^2}{s} + 1)(n_f + i)^2. \end{aligned}$$

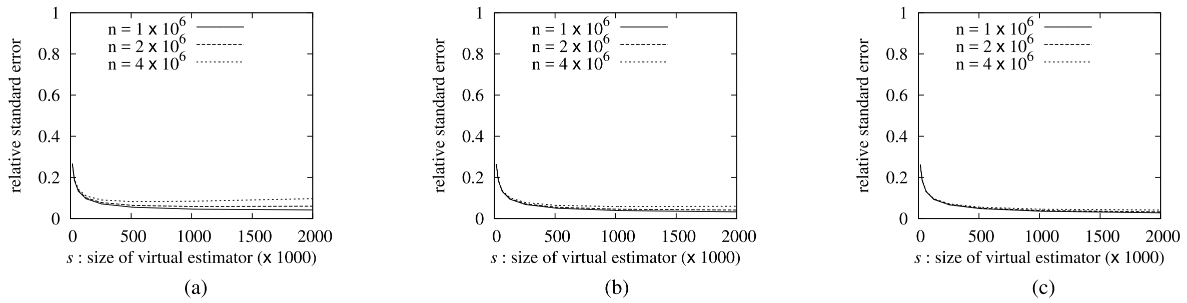


Fig. 10. Relative standard error of vHLL with respect to s, n, and n_f. (a) Flow cardinality n_f=1 $\times$ 10^4. (b) Flow cardinality n_f=2 $\times$ 10^4. (c) Flow cardinality n_f=4 $\times$ 10^4.

where β is equal to 1.04 for vHLL, and is equal to 0.78 for vPCSA. Combining (14) with the above equation, we have

$$\begin{aligned}&E(\hat{n_s}^2)\\&=\sum_{i=0}^{n-n_f}E(\hat{n_s}^2\mid n_s-n_f=i)\,Prob\{n_s-n_f=i\}\\&\approx\sum\nolimits_{i=0}^{n-n_f}(\frac{\beta^2}{s}+1)(n_f+i)^2\binom{n-n_f}{i}(\frac{s}{m})^i(1-\frac{s}{m})^{n-n_f-i}\\&=(\frac{\beta^2}{s}+1)\big(n_f{}^2+2n_fE(n_s-n_f)+E((n_s-n_f)^2)\big)\\&=(\frac{\beta^2}{s}+1)\big(n_f{}^2+2n_f(n-n_f)\frac{s}{m}\\&\qquad+(n-n_f)\frac{s}{m}(1-\frac{s}{m})+(n-n_f)^2(\frac{s}{m})^2\big)\\&=(\frac{\beta^2}{s}+1)\big((n_f+(n-n_f)\frac{s}{m})^2+(n-n_f)\frac{s}{m}(1-\frac{s}{m})\big).\end{aligned}\tag{25}$$

Applying (17), (24) and (25) to (19), we have

$$\begin{aligned}Var(\hat{n_f})\approx(\frac{m}{m-s})^2\Big(&\frac{\beta^2}{s}\big(n_f+(n-n_f)\frac{s}{m}\big)^2\\&+(n-n_f)\frac{s}{m}(1-\frac{s}{m})+(\frac{s}{m})^2\frac{\beta^2}{m}n^2\Big),\end{aligned}\tag{26}$$

where β is equal to 1.04 for vHLL, and to 0.78 for vPCSA.

Since 0.78 is a smaller value of β than 1.04, from (26), it may appear that vPCSA has smaller variance than vHLL, suggesting that vPCSA is more accurate. However, a PCSA register is 32 bits long, while a HLL register is only 5 bits long. A four-byte machine word can accommodate only one PCSA register, but it can accommodate six HLL registers. So when vHLL and vPCSA solutions are given the same amount of memory, the total number of registers or the m value for vHLL is six times larger than that of vPCSA, which will reduce the standard error of vHLL by 45%. Later in Section VIII-E, we will empirically compare the accuracy of vHLL and vPCSA.

We give an intuitive explanation to the three terms in (26) between the parentheses after $\big(\frac{m}{m-s}\big)^2$. We know that the noise $n_s - n_f$ in a virtual estimator follows a binomial distribution $Bino(n - n_f, \frac{s}{m})$, whose mean value is given by (3) as $(n - n_f)\frac{s}{m}$ and whose variance is $(n - n_f)\frac{s}{m}(1 - \frac{s}{m})$. The noise variance is captured by the second term in (26). The average number of elements a virtual estimator receives is $n_f + (n - n_f)\frac{s}{m}$, the flow cardinality plus the mean noise. So on average, the variance of a virtual estimator is $\frac{\beta^2}{s}\big(n_f + (n - n_f)\frac{s}{m}\big)^2$, which is the first term in (26). The third term $(\frac{s}{m})^2\frac{\beta^2}{m}n^2$ is caused by the estimation error of the grand flow cardinality n.

C. Relative Standard Error

We define the relative standard error as

$$StdErr(\frac{\hat{n_f}}{n_f})=\frac{\sqrt{Var(\hat{n_f})}}{n_f}.\tag{27}$$

From (26) and (27), we observe that the relative standard error (or error in short) increases as the grand-flow cardinality n increases, and it reduces as the target-flow cardinality n_f grows.

Below we use some numerical examples to illustrate the above observations and the interplay between different sources of estimation error. Suppose the allocated memory is m = 256K. Consider a target flow cardinality of n_f = 10^4. Figure 10(a) shows the estimation error of vHLL numerically computed from (27) with respect to s (the number of registers per virtual estimator) on the horizontal axis and n (the combined cardinality of all flows) for different curves. Starting from 16, as s increases, the error drops quickly, thanks to improved estimation accuracy from the virtual estimator M_f as predicted by Theorem 1. However, when s increases further (more than 256 in the figure), the rate of improvement drops significantly, which can also be predicted by Theorem 1 with its factor of improvement being $\frac{1}{\sqrt{s}}$. Moreover, as s increases, the error caused by noise increases. Combining these two factors, we observe that when s is relatively large (for a wide range from 500 to 2000 in the figure), its impact on the error becomes more or less stabilized.

From Figure 10(a) to Figure 10(c), we increase n_f and observe that the error decreases, which means that the *relative* standard error is smaller for flows of larger cardinalities (although their *absolute* errors can still be larger). When n increases, the error increases, as predicted.

VIII. Experimental Evaluation

We have implemented both vHLL and vPCSA algorithms, which are both based on register-sharing. We compare their performance through experiments using real network traces

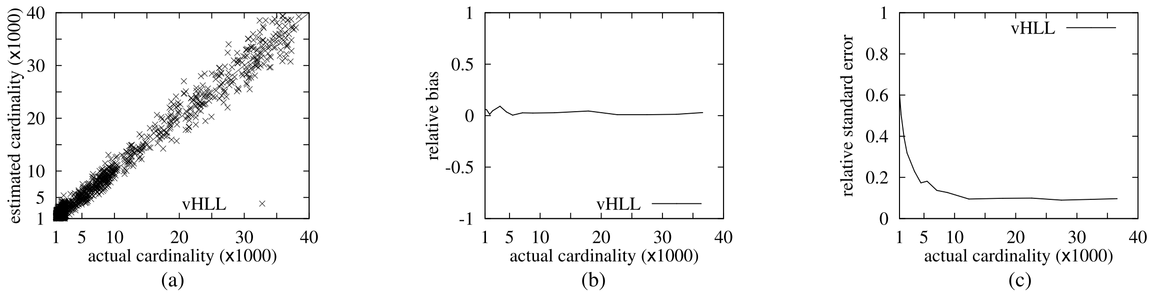


Fig. 11. Performance of vHLL with 0.25 bit memory per flow. (a) vHLL with 0.25 bit per flow. (b) Estimation bias. (c) Estimation accuracy.

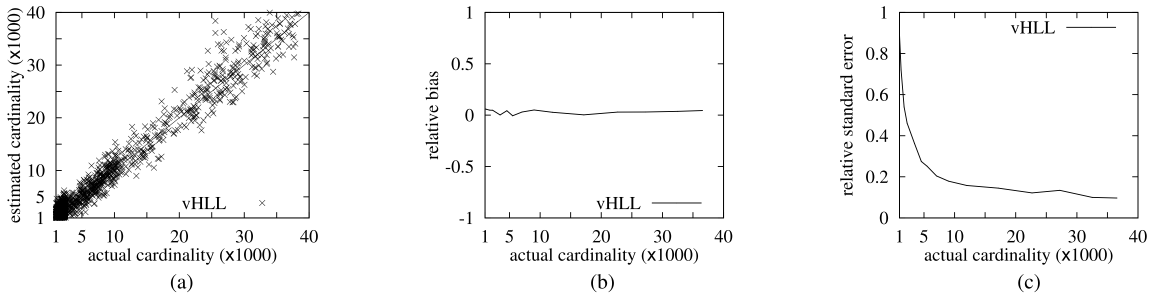


Fig. 12. Performance of vHLL with 0.1 bit memory per flow. (a) vHLL with 0.1 bit per flow. (b) Estimation bias. (c) Estimation accuracy.

downloaded from CAIDA [25]. The traces are captured by a high-speed monitor named equinix-sanjose (located in San Jose, CA, US), which is connected to a 10-Gbit/s Ethernet backbone link. Each trace file captures the packets in 1 minute. In order to create larger traces for our experiments, we download 60 traces and combine them into 6 larger ones, each for 10 consecutive minutes. The statistics of the large traces can be found in the following table.

time(min)	num of flows	total cardinality	avg flow cardinality
1-10	1473306	2675506	1.8
11-20	1013517	1856676	1.8
21-30	1648779	3005649	1.8
31-40	1562288	2881330	1.8
41-50	1612709	3280242	2.0
51-60	1612605	3280138	2.0

We consider per-source flows and measure the number of distinct destinations that each source sends packets to. The distribution of the flows with respect to the cardinality has been shown previously in Figure 6. We design the experiments primarily for evaluating how accurate the proposed new solutions of cardinality estimation are. We also include a case study on detecting super destinations.

Among the new solutions, vHLL outperforms vPCSA in space-accuracy tradeoff. In order to save space, we only include the results of vHLL for detailed evaluation in tight-memory setting, while providing the performance comparison between vPCSA and vHLL in the end.

A. Estimation Accuracy in Tight Memory

We evaluate the impact of memory space on the accuracy of cardinality estimation for vHLL. For the proposed vHLL, we configure the value of s to 512 by default, but will vary its value in later experiments. Recall that m is the total number of registers in the common pool. Its value depends on the overall available memory. The average number of flows in all six traces is about 1.5 millions. We vary the available memory space from 0.375 Mb to 0.15 Mb, such that the average memory per flow is about 0.25 bit and 0.1 bit, respectively. The corresponding experimental results are presented in Figures 11 and 12, respectively. Again, each flow is represented by a point, whose x-coordinate is the true cardinality and y-coordinate is the estimated cardinality. The equality line is also shown. The closer a point is to the line, the more accurate the estimation is.

In Figure 11, plot (a) shows the performance of vHLL with average memory of 0.25 bit per flow. The points are clustered around the equality line ($y = x$), indicating good accuracy. Plot (b) shows estimation bias. The vertical axis is the relative bias defined as $E(\frac{n_f - \hat{n_f}}{n_f})$. Since there are too few flows for some cardinalities (especially the large ones) in our Internet trace, we divide the horizontal axis into measurement bins of width from 5000 on the high end in the plots to 1000 in the low end to ensure that each bin has a sufficient number of flows 25, and measure the bias and standard deviation in each bin. Plot (c) shows estimation accuracy. The vertical axis is the relative standard error of the estimation results, which is defined as $\frac{\sqrt{Var(\hat{n_f})}}{n_f}$. The measurement also uses the bin method as previously explained.

As the average memory per flow decreases further to 0.1 bit, Figure 12 show that vHLL still works with gradually deteriorating accuracy. We also point out that although the relative standard errors for small flows are higher, it does not entirely diminish the usefulness of these estimations because the absolute errors for small flows are in fact much smaller than those of large ones. For example, by examining the first

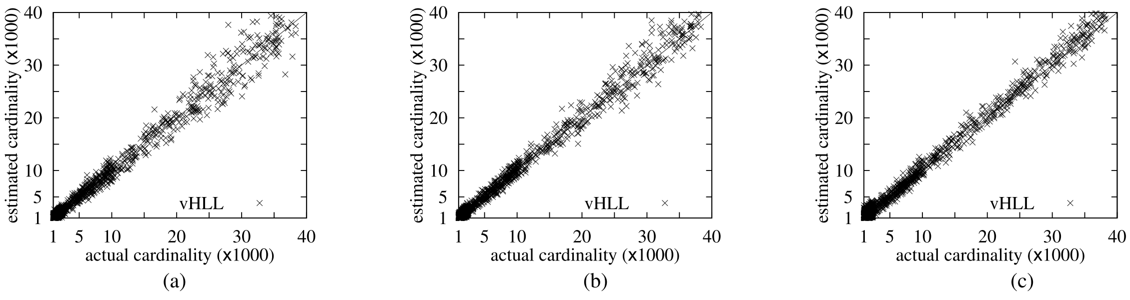


Fig. 13. Cardinality estimation with different values of s under average memory of 0.5 bit per flow ($m = 2^{17}$). (a) Virtual estimator size $s = 128$. (b) Virtual estimator size $s = 256$. (c) Virtual estimator size $s = 1024$.

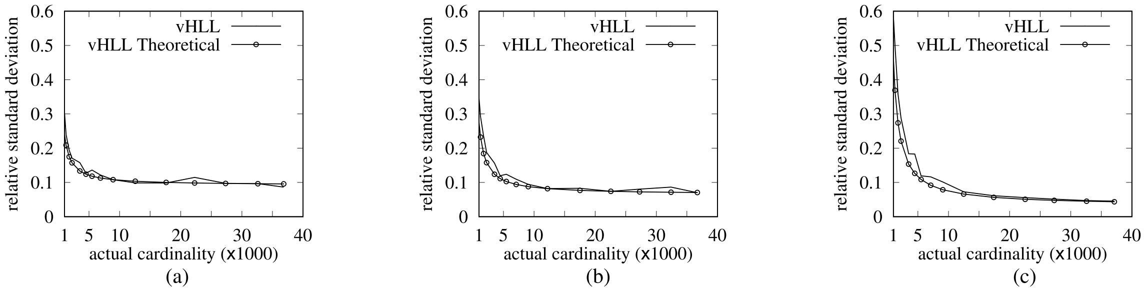


Fig. 14. Relative standard errors of cardinality estimation with different values of s under average memory of 0.5 bit per flow ($m = 2^{17}$). (a) Virtual estimator size $s = 128$. (b) Virtual estimator size $s = 256$. (c) Virtual estimator size $s = 1024$.

plot of each figure, one will not mistaken a small flow for a large one due to the modest absolute error.

B. Impact of Value s on vHLL

Our second set of experiments evaluate the impact of s (number of registers per virtual estimator) on estimation accuracy. We fix the average memory to 0.5 bit per flow, but change s from 512 to values: 128, 256 and 1024. The results are shown in Fig. 13(a)-(c), respectively. Corresponding relative standard errors are shown in Figure 14(a)-(c), respectively.

We observe that when s is relatively small at 128, the estimation accuracy in Figure 13(a) is noticeably worse than that in Figure 13(b), which is evident from the fact that the points of the former surround the equality line less tightly. Quantitatively, the errors in Figure 14(a) with $s = 128$ are larger than those in Figure 14(c) for vHLL with $s = 1024$. For example, when the actual cardinality is 20000, the relative standard error under $s = 128$ is 10.9%, while that under $s = 512$ is 6.5%.

However, when s becomes large enough (more than 256), for a wide range of values, the impact of s on the estimation accuracy stabilizes, which is evident when comparing Figure 13(b) and Figure 13(c), whose s values are 256 and 512, respectively. For example, when the actual cardinality is 20000, their errors are 8.1% and 6.5%, based on from Figures 14(b) and 14(c), respectively.

In Figure 14, we have also illustrated the theoretical standard deviation predicted by equations (27) and (26), where the error factor β is assigned to 1.04 for vHLL. So the above empirical observations are consistent with our analysis in Section VII and the numerical results in Fig. 10 (which has different parameters though). The reasons for these observations were explained in Section VII-C and will not be repeated here.

C. Impact of Overall Traffic

Our third set of experiments investigate how the overall traffic volume affects estimation accuracy. The overall traffic volume is characterized by n, the sum of all flows' cardinalities, because duplicates in the traffic must be removed in our context. The greater the value of n is, the larger the average noise level on each register will be, which will in turn negatively affect the estimation accuracy of a virtual estimator consisting of s registers.

We artificially increase the cardinality of each flow by a factor randomly chosen from the range of $[1, 3]$, which doubles the cardinality on average. The value of n is thus expected to be doubled. We then repeat the experiment in Figure 13 with average memory of 0.5 bit per flow. The results are presented in Figure 15, where plot (a) shows raw estimated cardinalities, plot (b) shows the estimation bias, and plot (c) shows the relative standard error. The bias remains close to zero, particularly for large flows. The error is modest, but larger than that in Figure 13(c) where the value of n is half, which confirms our prediction above.

We further enlarge n by increasing the cardinality of each flow with a factor randomly chosen from the range of $[1, 7]$. The value of n is expected to be increased by four folds. The results are presented in Figure 16. Again the bias is close to one, but the error increases.

D. A Case Study: Detect Super Destinations

Our fourth set of experiments applies vHLL to a hypothetical application for detecting so-called super destinations.

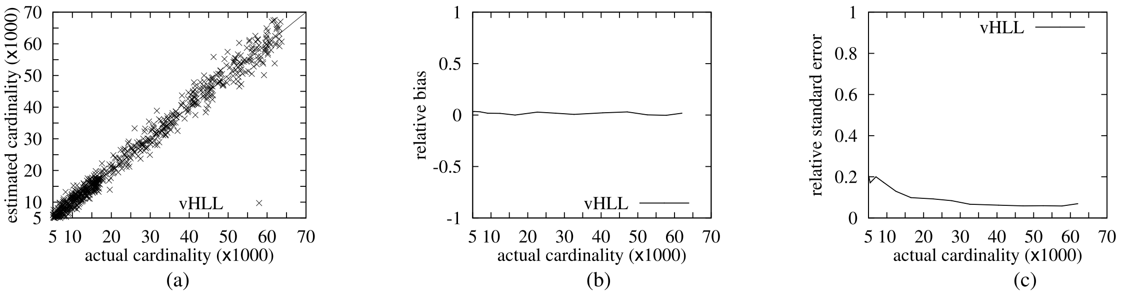


Fig. 15. Cardinality estimation with n doubled under average memory of 0.5 bits per flow. (a) n is doubled. (b) Estimation bias. (c) Estimation accuracy

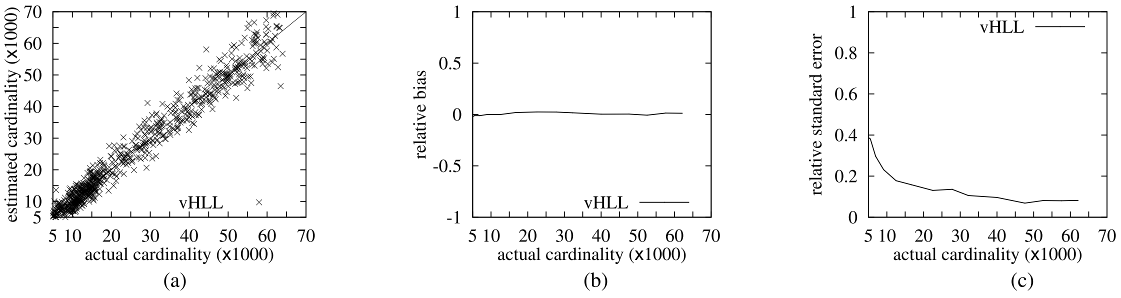


Fig. 16. Cardinality estimation with n increased four folds under average memory of 0.5 bits per flow. (a) n is increased by four folds. (b) Estimation bias. (c) Estimation accuracy.

TABLE II
FALSE POSITIVE RATIO AND FALSE NEGATIVE RATIO WITH RESPECT TO MEMORY COST

Memory (bit per flow)	vHLL	
	FPR	FNR
0.25	0.039	0.026
0.5	0.034	0.013
1	0.012	0.014

In this case study, we consider per-destination flows and measure the number of distinct sources that access a destination address in each measurement period, using the same Internet traces. Suppose the policy is to report all the destinations that have been accessed by 5,000 or more sources within a measurement period. These *super destinations* may be used for profiling the popular servers (or services) in the network or triggering anomaly warnings (such as DDoS attacks) if they were never reported as super destinations before.

If a destination with a cardinality less than 5,000 is reported, it is called a *false positive*. If a destination with a cardinality 5,000 or above is not reported, it is called a *false negative*. We define the false positive ratio (FPR) as the number of false positives divided by the total number of destinations reported. Based on this definition, if FRP is 0.1, it means 10% of the reported destinations should not have been reported. We define the false negative ratio (FNR) as the number of false negatives divided by the number of destinations whose cardinalities are 5,000 or more.

The experimental results are shown in Table II. vHLL has non-negligible FPR and FNR since its estimated cardinality is not exactly the true cardinality. To confine impreciseness to a certain degree, the policy may be relaxed to report all destinations whose estimated cardinalities are $5000 \times (1 - \epsilon)$ or above, where $0 \leq \epsilon < 1$. If a destination less than $5000 \times\ (1 - 2\epsilon)$ gets reported, it is called an ϵ-*false positive*. If a destination with a true cardinality 5,000 or more is not reported, it is called an ϵ-*false negative*. The FPR and FNR are defined the same as before. The experimental results for $\epsilon = 10\%$ are shown in Table III, and those for $\epsilon = 20\%$ are shown in Table IV, where the FPR and FNR for vHLL are merely 0.7% and 0.6%, respectively, when the memory is 0.25 bit per flow. In Table IV, when the memory grows to at least 0.5 bit per flow, FPR and FNR for vHLL become zeros.

TABLE III
$\epsilon = 10\%$, FALSE POSITIVE RATIO AND FALSE NEGATIVE RATIO WITH RESPECT TO MEMORY COST

Memory (bit per flow)	vHLL	
	FPR	FNR
0.25	0.014	0.010
0.5	0.003	0.003
1	0.003	0.002

TABLE IV
$\epsilon = 20\%$, FALSE POSITIVE RATIO AND FALSE NEGATIVE RATIO WITH RESPECT TO MEMORY COST

Memory (bit per flow)	vHLL	
	FPR	FNR
0.25	0.007	0.006
0.5	0.0	0.0
1	0.0	0.0

E. Comparison Study of vHLL and vPCSA

In the last set of experiments, we evaluate the performance of vHLL and vPCSA together to demonstrate the broad

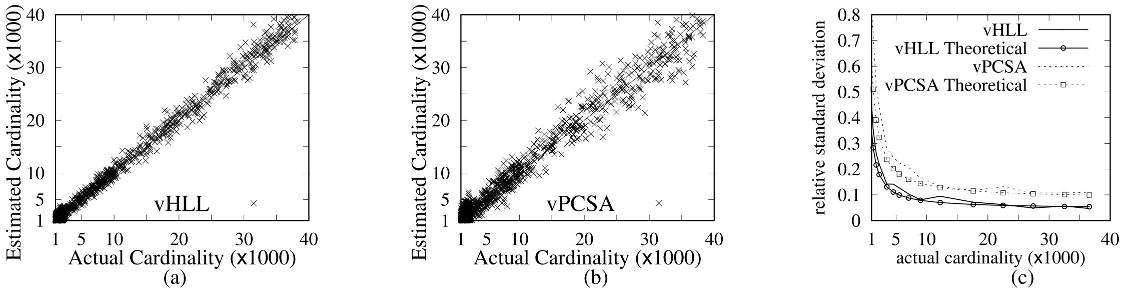


Fig. 17. Compare vHLL and vPCSA when they are given the same amount of memory of 0.5 bits per flow. (a) vHLL with $s = m/2^8$. (b) vPCSA with $s = m/2^8$. (c) Comparison of estimation accuracy.

applicability of our virtual estimator framework on different types of estimators. In Figure 17(a) and (b), we present the estimation accuracy of vHLL and vPCSA, respectively, under the same amount of memory, 0.5 bit per flow. The two plots show that both solutions can generate good-quality cardinality estimations for elephant flows.

In Figure 17(c), we present both the empirical standard deviation of the estimation results in the previous two plots and the theoretical standard deviation predicted by equations (27) and (26), where the error factor β is replaced by 1.04 for vHLL and by 0.78 for vPCSA. The empirical results and the theoretical results are consistent, both showing that the standard deviation of vHLL is 45% smaller than that of vPCSA.

IX. Conclusion

In this paper, we have proposed a unified framework for developing efficient solutions to the problem of estimating cardinalities for a very large number of streaming flows. From this framework, we examine two new solutions in details, including a particularly powerful solution called virtual HyperLogLog (vHLL). Through analysis and experimental evaluation, we show that vHLL can use a compact memory space (down to 0.1 bit per flow on average) to estimate the cardinalities of flows with wide range and reasonable accuracy. This new capability enables on-chip implementation of cardinality estimation needed for online applications that can keep up with the line speed of modern routers, or allow efficient processing of big data by using low-cost commodity computers instead of expensive high-performance computing systems.

References

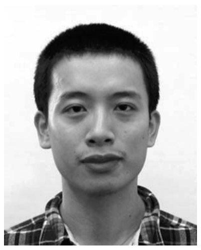
[1] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proc. ACM SIGCOMM*, Aug. 2002, pp. 270–313.
[2] Q. Zhao, J. Xu, and A. Kumar, "Detection of super sources and destinations in high-speed networks: Algorithms, analysis and evaluation," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 10, pp. 1840–1852, Oct. 2006.
[3] M. Yoon, T. Li, S. Chen, and J.-K. Peir, "Fit a spread estimator in small memory," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 504–512.
[4] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high-speed links," *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, pp. 925–937, Oct. 2006.
[5] P. Lieven and B. Scheuermann, "High-speed per-flow traffic measurement with probabilistic multiplicity counting," in *Proc. IEEE INFOCOM*, Mar. 2010, pp. 1–9.
[6] S. Heule, M. Nunkesser, and A. Hall, "HyperLogLog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm," in *Proc. EDBT*, 2013, pp. 683–692.
[7] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *J. Comput. Syst. Sci.*, vol. 31, no. 2, pp. 182–209, 1985.
[8] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Trans. Database Syst.*, vol. 15, no. 2, pp. 208–229, Jun. 1990.
[9] Z. Bar-Yossef, T. S. Jayram, R. Kumar, D. Sivakumar, and L. Trevisan, "Counting distinct elements in a data stream," in *Proc. Int. Workshop Randomization Approx. Tech. Comput. Sci.*, 2002, pp. 1–10.
[10] M. Durand and P. Flajolet, "Loglog counting of large cardinalities," in *Proc. Eur. Symp. Algorithms*, 2003, pp. 605–617.
[11] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier, "HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm," in *Proc. Int. Conf. Anal. Algorithms*, 2007, pp. 137–156.
[12] W. D. Gardner, "Researchers transmit optical data at 16.4 Tbps," *Inf. Week*, Feb. 2008. [Online]. Available: https://www.informationweek.com/researchers-transmit-optical-data-at-164-tbps/d/d-id/1065090
[13] *Google Trends*. Accessed: 2017. [Online]. Available: http://www.google.com/trends/
[14] T. Li, S. Chen, W. Luo, M. Zhang, and Y. Qiao, "Spreader classification based on optimal dynamic bit sharing," *IEEE/ACM Trans. Netw.*, vol. 21, no. 3, pp. 817–830, Jun. 2013.
[15] T. Li, S. Chen, and Y. Ling, "Fast and compact per-flow traffic measurement through randomized counter sharing," in *Proc. INFOCOM*, Apr. 2011, pp. 1799–1807.
[16] G. Cormode and S. Muthukrishnan, "An improved data stream summary: The Count-Min sketch and its applications," in *Proc. Latin Amer. Symp. Theor. Inform.*, 2004, pp. 29–38.
[17] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *International Colloquium on Automata, Languages, and Programming*. London, U.K.: Springer-Verlag, Jul. 2002. [Online]. Available: https://dl.acm.org/citation.cfm?id=684566
[18] Y. Lu and B. Prabhakar, "Robust counting via counter braids: An error-resilient network measurement architecture," in *Proc. INFOCOM*, Apr. 2009, pp. 522–530.
[19] X. Dimitropoulos, P. Hurley, and A. Kind, "Probabilistic lossy counting: An efficient algorithm for finding heavy hitters," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 1, p. 5, 2008.
[20] Q. Xiao, Y. Qiao, M. Zhen, and S. Chen, "Estimating the persistent spreads in high-speed networks," in *Proc. IEEE ICNP*, Oct. 2014, pp. 131–142.
[21] Q. Xiao, B. Xiao, and S. Chen, "Differential estimation in dynamic RFID systems," in *Proc. INFOCOM*, Apr. 2013, pp. 295–299.
[22] P. Indyk and D. Woodruff, "Tight lower bounds for the distinct elements problem," in *Proc. IEEE FOCS*, Oct. 2003, pp. 283–288.
[23] K. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla, "On synopses for distinct-value estimation under multiset operations," in *Proc. ACM SIGMOD*, 2007, pp. 199–210.
[24] N. Ntarmos, P. Triantafillou, and G. Weikum, "Counting at large: Efficient cardinality estimation in Internet-scale data networks," in *Proc. ICDE*, Apr. 2006, p. 40.
[25] *CAIDA UCSD Anonymized 2013 Internet Traces-January 17*. Accessed: 2013. [Online]. Available: http://www.caida.org/data/passive/passive_2013_dataset.xml
[26] B. Scheuermann and M. Mauve, "Near-optimal compression of probabilistic counting sketches for networking applications," in *Proc. ACM Dial M-POMC (Workshop Found. Mobile Comput.)*, 2007, pp. 1–7.

Qingjun Xiao (M'12) received the B.Sc. degree from the Computer Science Department, Nanjing University of Posts and Telecommunications, China, in 2003, the M.Sc. degree from the Computer Science Department, Shanghai Jiao Tong University, China, in 2007, and the Ph.D. degree from the Computer Science Department, Hong Kong Polytechnic University, in 2011. He joined Georgia State University and the University of Florida, and was a Post-Doctoral Researcher for three years. He is currently an Assistant Professor with Southeast University, China. His research interests include protocol and algorithm design in network traffic measurement, wireless sensor networks, and RFID systems. He is a member of ACM.

Shigang Chen (M'02–SM'12–F'16) received the B.S. degree in computer science from the University of Science and Technology of China, Hefei, China, in 1993, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana–Champaign, USA, in 1996 and 1999, respectively. He was with Cisco Systems, San Jose, CA, USA, for three years and the University of Florida, Gainesville, FL, USA, in 2002, where he is currently a Professor with the Department of Computer and Information Science and Engineering. He served on the Technical Advisory Board for Protego Networks from 2002 to 2003. He published over 100 peer-reviewed journal/conference papers. He holds 11 U.S. patents. His research interests include computer networks, Internet security, wireless communications, and distributed computing.

Dr. Chen received the IEEE Communications Society Best Tutorial Paper Award in 1999 and the NSF CAREER Award in 2007. He served in the steering committee of the IEEE IWQoS from 2010 to 2013. He is currently an Associate Editor of the IEEE TRANSACTIONS ON MOBILE COMPUTING. He was an Associate Editor for the other journals, such as the IEEE/ACM TRANSACTIONS ON NETWORKING and IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY.

You Zhou received the B.S. degree from the Department of Electronic Information Engineering, University of Science and Technology of China, Hefei, China, in 2013, He currently pursuing the Ph.D. degree in computer and information science and engineering at the University of Florida, Gainesville, FL, USA. His advisor is Prof. S. Chen. His research interests include network security and privacy, big network data, and Internet of Things.



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

Junzhou Luo received the B.S. degree in applied mathematics and the M.S. and Ph.D. degrees in computer network from Southeast University, Nanjing, China, in 1982, 1992, and 2000, respectively. He is a Full Professor with the School of Computer Science and Engineering, Southeast University. He is a member of the IEEE Computer Society and Co-Chair of the IEEE SMC Technical Committee on Computer Supported Cooperative Work in Design. He is a member and the Chair of ACM SIGCOMM China. His research interests are future network architecture, network security, cloud computing, and wireless LAN.

Tengli Li is currently pursuing the B.E. degree with the School of Computer Science and Engineering, Southeast University, Nanjing, China. She is currently a member with the Research Team lead by Dr. Q. Xiao, and is responsible for building test bed to monitor network traffic traversing virtual switch and server. Her research interests include next-generation network architecture, big network data, and network security.

Yibei Ling (M'00–SM'06) received the B.S. degree in electrical engineering from Zhejiang University, Hangzhou, China, in 1982, the M.S. degree in biostatistics from Shanghai Medical University, Shanghai, China, in 1985, and the Ph.D. degree in computer science from Florida State University, Miami, FL, USA, in 1995.

He is currently a Senior Research Scientist with the Applied Research Laboratories, Telcordia Technologies, Morristown, NJ, USA, where he is developing wireless middleware systems and wireless messaging systems. He has published several papers in the IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, the IEEE TRANSACTIONS ON BIOMEDICAL ENGINEERING, *SIGMOD*, *Data Engineering*, *Information Systems*, and *Operating Systems Review*. His research interests include distributed systems, system performance evaluation, query optimization in database management systems, and biological modeling.