# Highly Compact Virtual Counters for Per-Flow Traffic Measurement through Register Sharing

You Zhou[†] Yian Zhou[†] Min Chen[†] Qingjun Xiao[‡] Shigang Chen[†]
[†]Department of Computer & Information Science & Engineering, University of Florida, Gainesville, FL, USA
[‡]Key Lab of Computer Network and Information Integration, Southeast University, Ministry of Education, China
Email: {youzhou, yian, min}@cise.ufl.edu csqjxiao@seu.edu.cn sgchen@cise.ufl.edu

*Abstract*—**Per-flow traffic measurement is a fundamental problem in the era of big network data, providing critical information for many practical applications including capacity planning, traffic engineering, data accounting, resource management, and scan/intrusion detection in modern computer networks. It is challenging to design highly compact data structures for approximate per-flow measurements. In this paper, we show that a highly compact virtual counter architecture can achieve fast processing speed (slightly more than 1 memory access per packet) and provide accurate measurement results under tight memory allocation. Extensive experiments based on real network trace data demonstrate its superior performance over the best existing work.**

## I. Introduction

There is hardly any other data set whose size can rival the big network data that flows on the Internet. The annual global IP traffic is expected to pass zettabyte by 2016 [1]. As we move into the big network data era, one fundamental problem is per-flow traffic measurement [2]–[6], which is to count the *number of packets in each flow* (or called *flow size*). The flow is uniquely identified by a particular field in the packet header called *flow label*, which can be flexibly defined depending on application context. The flows under measurement may be per-source flows (flow label is source address), per-destination flows, per-source/destination flows, TCP flows, or other user-defined flows.

Per-flow traffic monitoring and measurement have many important applications. We may measure the number of packets in each TCP flow, the data volume of each voice-over-IP session, the number of bytes that each host downloads, the number of SYN packets sent from each source address, or the number of SYN-ACK packets sent to each address. Such per-flow data are helpful to provide important information for capacity planning, traffic engineering, data accounting, resource management, and scan/intrusion detection in modern computer network [7]–[16]. For example, measuring the number of SYN-ACK packets provides a means to detect SYN attacks or worm scanning [13] [16]. For another example, Internet service provider can combine the per-flow traffic information to align traffic distribution in the network, and help discover network traffic patterns and reduce the congestion.

Modern routers forward packets from incoming ports to outgoing ports via switching fabric. To process packets in real time, online modules for traffic measurement, packets scheduling, access control, and quality of service are implemented on network processors, bypassing main memory and CPU almost entirely. To keep up with the line speed of modern routers, the per-flow measurement module should minimize the processing time per packet, and needs to be designed in high speed but expensive on-die SRAM [5] [6]. The limited SRAM size poses major challenge for per-flow traffic measurement.

It has been demonstrated that giving each flow a counter in SRAM cannot scale for today's big network data [7]. In addition, exact counting for each flow is also not practical due to large memory and computation overhead. Therefore, approximate estimation that can provide probabilistic guarantees becomes the only viable option. The representative state-of-art estimation methods include Counter Braids [3] [4], randomized counter sharing [5], and Counter Tree [6].

Counter Braids (CB) [3] [4] gives accurate per-flow measurement. It reduces memory overhead by sharing counters among flows. The counters are arranged in two or more levels, and each flow is mapped to $k$ counters at every level. When there are two levels and $k = 3$, CB performs 6 (occasionally 12) memory accesses to encode one packet, which limits its online speed.

Li et al. [5] proposed a counter sharing architecture called randomized counter sharing. Each flow is recorded by hundreds of counters, and flows share their counters randomly from a common counter pool. It takes 2 memory accesses and 1 hash computation to encode each packet. Its major drawback is that the estimation range is limited [6].

Chen et al. [6] proposed a two-dimensional counter sharing architecture called Counter Tree (CT); not only do flows share their counters, but the counters share their high-order bits based on a tree structure. It achieves better memory efficiency than all previous work. Nevertheless, CT still requires at least 2 bits per flow in memory consumption and slightly more than 2 memory accesses per packet in processing time. Our experiments based on real network trace data show that CT cannot work well under a tight memory space, e.g., less than 1 bit per flow. Other approaches employ an online sampling module where each arriving packet is sampled with a probability before being encoded to a counter. However, [2] demonstrates that aggressive sampling introduces significant error, especially for small-size flows.

This paper presents a novel counter architecture for per-flow traffic measurement that can further improve the memory and processing time efficiency as well as measurement accuracy. It is a highly compact and fast architecture, called Virtual HyperLogLog Counter (VHC), which achieves faster processing time than the prior of art. To encode a packet, VHC only requires slightly more than 1 memory access and can work under a tight memory where the previous best approach [6] does not perform well.

The rest of the paper is organized as follows. Section II defines the performance metrics. Section III describes the design of virtual counters. Section IV presents the online encoding and offline estimation modules of VHC. Section V gives the experiment results. Section VI draws the conclusion.

## II. Performance Metrics

In this paper, we employ the following three metrics same as [5] and [6] to evaluate the performance of a per-flow traffic measurement scheme.

*1) **Processing time**:* The average time required for encoding a packet, particularly measured by the average number of memory accesses and the number of hash value computations as in [5] and [6]. In order to keep up with the line speed of modern routers, the processing time for encoding a packet should be made as small as possible.

*2) **Memory requirement**:* The minimal memory (in the sequel memory refers to SRAM) required to achieve reasonably sound measurement results for per-flow traffic measurement. The great disparity in memory demand and supply requires us to implement as compact as possible online per-flow traffic measurement modules. Hence we mainly focus on the memory requirement for implementing compact counter architectures.

*3) **Estimation accuracy**:* Let $n$ be the actual size of a flow, and $\hat{n}$ be the estimation result given by the measurement scheme. We evaluate the estimation accuracy by the relative bias $Bias(\frac{\hat{n}}{n})$ and relative standard error $StdErr(\frac{\hat{n}}{n})$, defined below as [6]:

$$\begin{aligned} Bias(\frac{\hat{n}}{n}) &= E(\frac{\hat{n}}{n}) - 1, \\ StdErr(\frac{\hat{n}}{n}) &= \sqrt{Var(\frac{\hat{n}}{n})} = \frac{\sqrt{Var(\hat{n})}}{n}. \end{aligned} \quad (1)$$

Clearly, smaller values of relative bias and relative standard error represent more accurate measurement results. Given a certain available memory space, the estimation results should be made as more accurate as possible.

## III. Design of Virtual Counters

### A. Motivation

According to the study in [17], 9% of the flows account for 90% of the Internet traffic. Without loss of generality, we use the real network trace captured by the main gateway of our university as an example, which contains about 68 million TCP flows and 750 million packets. The flow size distribution is illustrated in Figure 1 in log scale, where each point represents the number ($y$-coordinate) of flows that have a particular size ($x$-coordinate). Clearly, the vast majority of flows have small sizes, while only a small number of flows have large sizes.

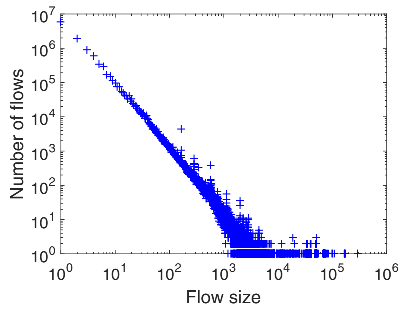

Fig. 1: Flow size distribution. Each point represents the number ($y$-coordinate) of flows that have a certain size ($x$-coordinate).

With a large number of flows, it is not advisable to maintain one counter for each flow, given the limited size of SRAM. The reason is that, when we don't know the flow sizes in advance (which are in fact what we want to figure out), the sizes of all counters should be set according to the 'elephant' flows that have large flow sizes. To achieve reasonably accurate measurement results for 'elephant' flows, each counter may need to be as large as 32 bits. This translates to a total memory consumption of as much as 272 MB for the 68 million TCP flows, which is obviously not acceptable. If we take a closer analysis, we can observe that for the majority of flows that have small sizes, the high-order bits in their counters are actually under-utilized as many or even most of them remain zeros. To improve the memory efficiency, counter sharing should be enabled among the flows to utilize these unused bits.

### B. Counter Sharing

A common counter sharing mechanism is illustrated in Figure 2, where each cell represents a base counter. Each flow randomly picks a number of base counters from the physical counter array to form its virtual counter. Since the virtual counters of all flows share the same base counter pool (physical counter array), large flows can 'borrow' memory from small flows to utilize the under-utilized base counters.

Li et at. [5] adopt this basic idea to propose their counting architecture with one dimensional base-counter sharing. However, since small flows dominate all network flows, many high-order bits of the base counters are still under-utilized. If we reduce the number of high-order bits, the measurement range is also reduced. To construct more compact base counters while allowing a large measurement range, Chen et al. [6] propose a two-dimensional counter sharing idea, which allows base counters to share high-order bits. The memory efficiency is improved, but it introduces more noises among base counters, which leads to inaccurate estimations when the memory space

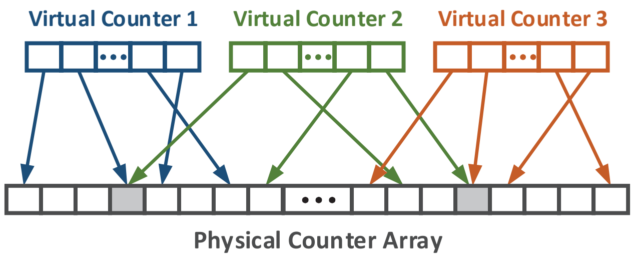

Fig. 2: An illustration of counter sharing.

is tight. We will explain more in Section V.

### C. *Register Sharing*

There is a related branch of research for measuring flow cardinality [18]–[20], which is defined as the number of distinct elements carried by a flow, where elements can be defined based on application needs. The HyperLogLog (HLL) sketches [20] use five-bit registers with an estimation range of up to $10^9$. A recent work on virtual HyperLogLog sketches shares registers among the flows to reduce the memory consumption [21]. We have two goals in this paper. First, we want to see if the idea of [21] can be adapted for a new virtual HyperLogLog counter architecture (VHC) that measures flow size (i.e., number of packets in each flow). Second, we perform experiments to evaluate how well such a counter architecture performs in comparison with the best existing flow-size estimator.

## IV. Virtual HyperLogLog Counter Architecture

In this section, we first describe the design of a VHC counter architecture, and then analyze its measurement accuracy. VHC includes an online encoding module and an offline estimation module. The online encoding module records the packets to the counter architecture in real time, while the offline estimation module measures the sizes of all flows based on the counter data recorded from online encoding.

### A. *Online Encoding*

In the online encoding module, a register array $C$ of $m$ HLL registers is used to store the packet information of all flows. For an arbitrary flow $f$, where $f$ is the flow label, we pseudo-randomly select $s$ registers from $C$ to logically form a virtual counter $C_f$, and use $C_f$ to encode the packets in flow $f$. Denote the $i$th register in the $C_f$ as $C_f[i]$, $0 \le i < s$. One way of selecting registers in $C$ to form $C_f$ is

$$C_f[i] = C[H(f \oplus H(i)) \mod m]\ ,\ 0 \le i < s, \tag{2}$$

where $H(\cdot)$ is a hash function whose range is $[0, m)$, and $\oplus$ is the XOR operator. We stress that $C_f$ will not be explicitly constructed in online encoding. The values of $H(i)$, $0 \le i < s$, can be pre-computed.

At the beginning of each measurement period, all registers in $C$ are initialized to zeros. When a packet arrives, the router extracts its flow label $f$ from its header, generates a pseudo-random number $q$ to select a register $C_f[q \mod s] = C[H(f \oplus H(q \mod s)) \mod m]$, and updates the value of this register as follows: Generate another pseudo-random binary number $q'$. Let $\lambda$ be the index of the leftmost 1 in $q'$. Namely, $q'$ is equal to the number of leading zeros in $q'$. For example, if $q' = 001\ldots$, then $\lambda = 2$. To update the value of the register, the router performs the following HyperLogLog operation [20]:

$$C[p] = \max\big(C[p], \lambda + 1\big), \tag{3}$$

where $p = H(f \oplus H(q \mod s)) \mod m$.

Hence, to encode a packet, the router only needs to calculate one hash function, generate two pseudo-random numbers, and perform at most two memory accesses: reading $C[p]$ and writing $C[p]$ back if its value changes. Generally, the probability to update the register $C[p]$ is relatively small (approaching to zero) when the total number of packets mapped to this register is large. Therefore, the processing time for each packet will be slightly more than 1 memory access on average. We will explain more in Section V-B with experiment results.

### B. *Offline Estimation*

At the end of each measurement period, the register array $C$ is offloaded to a server for long-term storage and offline query. Consider an arbitrary flow $f$ under offline query. We reconstruct its virtual counter $C_f$, where $C_f[i] = C[H(f \oplus H(i)) \mod m]$, $0 \le i < s$. Let $n_s$ be the number of packets encoded by $C_f$. The registers in $C_f$ form the HyperLogLog sketches that record the $n_s$ packets. Hence, we can estimate the value of $n_s$ from $C_f$ using the HyperLogLog formula [20]; let $\hat{n}_s$ be the estimated value. Due to register sharing, we know that $n_s$ is flow $f$'s size $n$ plus the noise $e$ introduced by other flows. Hence,

$$n = n_s - e \approx \hat{n}_s - e. \tag{4}$$

The average noise per register over the whole array $C$ is $\frac{N-n}{m} \approx \frac{N}{m}$, where $N$ is the total size of all flows. We can treat $C$ as the HyperLoglog sketches that record all $N$ packets and therefore estimate the value of $N$ from $C$ using the HyperLogLog formula [20]; let $\hat{N}$ be the estimated value of $N$. Therefore, we have

$$e \approx s\frac{\hat{N}}{m}. \tag{5}$$

Applying (5) in (4), we have

$$n \approx \hat{n}_s - s\frac{\hat{N}}{m}. \tag{6}$$

## V. Experiments

### A. *Experiment Setup*

We implement VHC as well as the state of art CT, and compare them through extensive experiments using real network traffic traces. The network traffic traces we use were collected by Cisco' NetFlow at the main gateway of our university.

The flows in the experiments can be per-source flows, per-destination flows and TCP flows, which all lead to the similar results. Without loss of generality, we use TCP flows for presentation. The network trace we use contains about 68 million TCP flows and 750 million packets. The trace segment used for our experiment contains $126,569,701$ packets which are generated by $11,453,043$ flows. The average flow size is $11.05$ packets/flow.

The performance metrics in Section II are used in our experiment evaluation, including per-packet processing time, memory requirement, and estimation accuracy. We run two sets of experiments to evaluate these metrics. The first set is used to evaluate the impact of memory size on the measurement accuracy of CT and VHC. We vary the available memory space $M$ from 0.25MB, 0.5MB, 1MB to 2MB, which translates to approximately 0.2bits/flow, 0.4bits/flow, 0.8bits/flow and 1.6bits/flow, respectively. To make a fair comparison, CT and VHC are given the same memory space to process the TCP traffic trace in each case. For CT, we implement a Counter Tree architecture with degree fixed to 3, virtual counter size fixed to 100, and node counter size fixed to 4 bits as Chen et al. [6] did in their experiments. The second set of experiments evaluates the impact of the virtual counter size $s$ on the performance of VHC. We fix the memory space $M$ = 1MB, and vary $s$ with different values to observe its estimation accuracy.

## B. Processing Time

In the first set of experiments, we record the average number of memory accesses and hash computations for encoding a packet by CT and VHC. The comparison results are presented in Table I. Both CT and VHC only need 1 hash computation for each packet to allocate its corresponding counter, which are quite efficient. In addition, CT needs more than 2 memory accesses to encode a packet. [6] gives an upper bound of amortized number of memory accesses $2 + \frac{1}{2^b+1}$, where $b$ is the size of node counter. When $b = 4$, the upper bound equals to 2.13. By contrast, VHC only requires slightly more than 1 memory access to encode a packet as shown in the table. This is consistent to our previous analysis. Recall from the online encoding module in Section IV-A, VHC only requires 1 memory access to read the value in the register, and only needs to write it back with a very small probability. Moreover, the average number of memory accesses of VHC decreases when less memory are available since each register is shared by more flows, which reduces the updating frequency. Clearly, VHC is more efficient than CT in terms of processing time.

## C. Estimation Accuracy and Memory Overhead

We study the estimation accuracy of CT and VHC when the available memory ranges from 0.25MB, 0.5MB, 1MB to 2MB. The experiment results of CT and VHC are presented in Figure 3 and Figure 4, respectively. Each figure includes four plots under different memory sizes $M$. Each point in each plot represents a flow, where the $x$ coordinate is the actual flow size $n$ and the $y$ coordinate is the estimated flow size $\hat{n}$. The equality line, $y = x$, is also shown. Clearly, the closer a point is to the equality line, the more accurate the estimate is.

TABLE I: Comparison of processing time by CT and VHC.

| memory size (MB) | number of memory accesses | | number of hash computaions | |
|---|---|---|---|---|
| | CT | VHC | CT | VHC |
| 0.25 | 2.13 | 1.02 | 1 | 1 |
| 0.5 | 2.13 | 1.03 | 1 | 1 |
| 1 | 2.12 | 1.06 | 1 | 1 |
| 2 | 2.11 | 1.10 | 1 | 1 |

In Figure 3, the first plot shows when the available memory is tight $M$ = 0.25MB (0.2bit/flow), CT gives meaningless results for majority of flows. As the memory size increases from 0.25MB to 1MB as shown in the second and third plots, CT becomes more clustered, but still cannot yield reasonable estimates. When the available memory space increases to 2MB, the fourth plot shows the points are clustered to the equality line, which indicates acceptable estimates. Figure 4 shows the experiment results of VHC. Clearly, VHC can generate very accurate estimates for both small and large flows as most points closely follow the equality line for all four plots. This is true even under a tight memory, e.g., $M$ = 0.25MB (0.2bit/flow) as the first plot shows. Moreover, through register sharing, VHC can easily handle wide counting ranges without modifying preset parameters, which is required by [6] in order to generate sound measurement results when facing different traffic situations. Therefore, VHC provides a more robust and flexible solution for real-life network traffic measurement.

The relative bias $Bias(\frac{\hat{n}}{n})$ and relative standard error $StdErr(\frac{\hat{n}}{n})$ of CT and VHC are given in Figure 5 and Figure 6, each of which includes four plots with a different size of memory $M$. From these two figures, we can see that both CT and VHC become more accurate when more memory space is used. It is also clear that VHC has smaller relative bias and relative standard errors than CT, which demonstrates VHC is indeed more accurate than CT.

## D. Impact of Value $s$

Our second set of experiments study the impact of the virtual counter size $s$ on the performance of VHC. We fix the available memory size to 1MB, and vary the value of $s$ from 512 to 128, 256 to 1024. The results are represented in Figure 7. The first three plots are estimation results under $s = 128$, 256 and 1024. Corresponding relative standard errors are illustrated in the fourth plot. Clearly, when $s$ is relatively small ($s$ = 128), the relative standard errors are larger than when $s$ = 256 or $s$ = 512 for large size flows. However, when $s$ gets large enough ($s = 1024$), the estimation accuracy for large size flows stabilizes, but the estimation accuracy for small size flows becomes noticeably worse. Combining these two effects, in practice, it may be more appropriate to choose a virtual counter size of either 256 or 512.

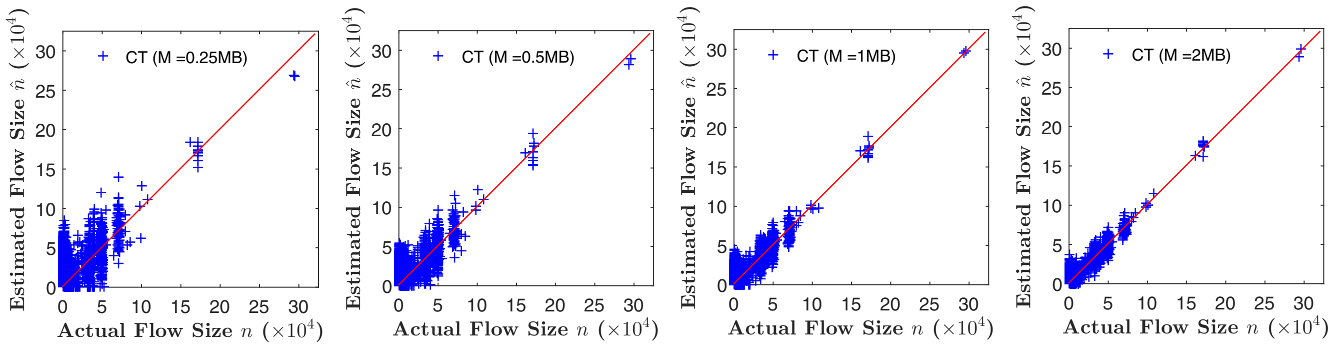

Fig. 3: Estimation results by CT. From left to right, memory size $M$ = 0.25MB, 0.5MB, 1MB, and 2MB.

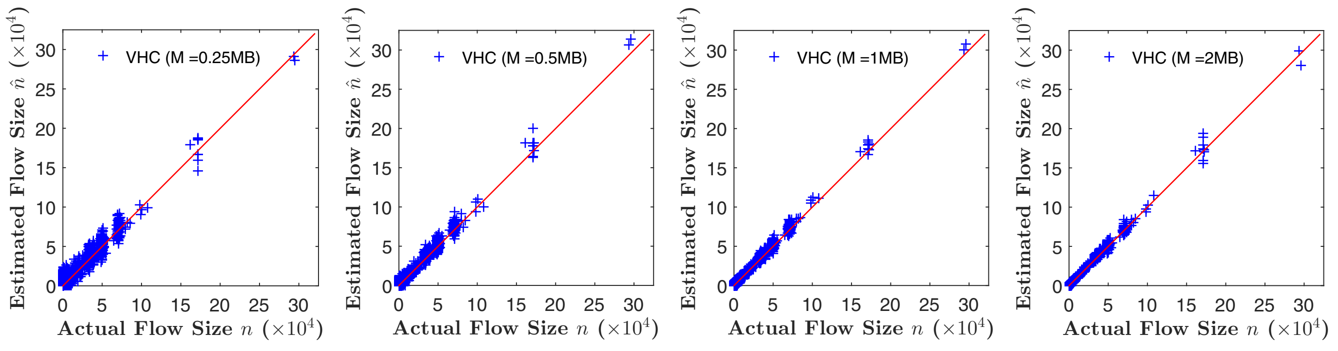

Fig. 4: Estimation results by VHC with $s = 512$. From left to right, memory size $M$ = 0.25MB, 0.5MB, 1MB, and 2MB.

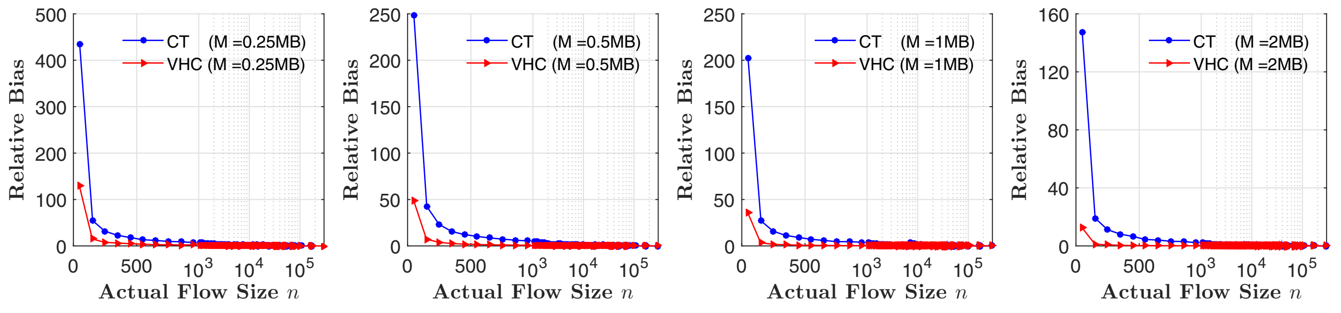

Fig. 5: Relative bias $Bias(\frac{\hat{n}}{n})$. From left to right, memory size $M$ = 0.25MB, 0.5MB, 1MB, and 2MB.

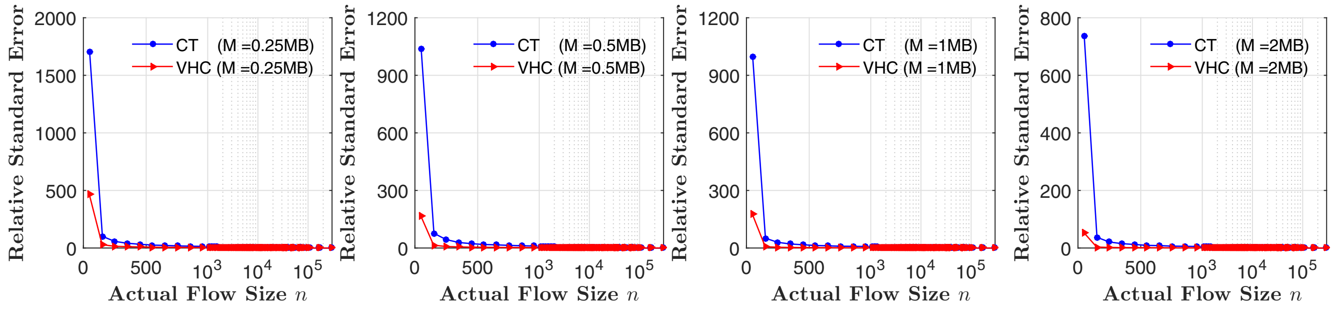

Fig. 6: Relative standard error $StdErr(\frac{\hat{n}}{n})$. From left to right, memory size $M$ = 0.25MB, 0.5MB, 1MB, and 2MB.

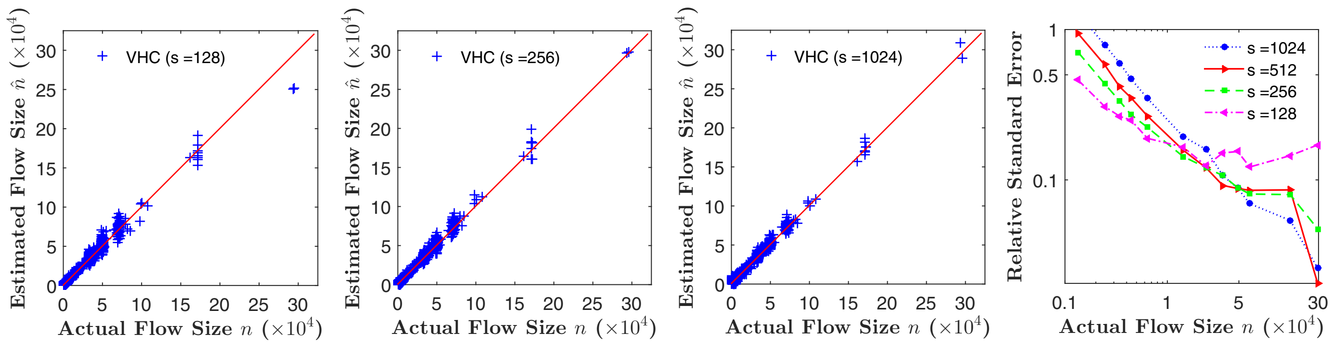

Fig. 7: Estimation results and relative standard errors of VHC under different value of $s$. Memory size $M$ = 1MB.

## VI. Conclusion

In this paper, we present a highly compact and fast counter architecture for per-flow traffic measurement, called Virtual HyperLogLog Counter (VHC), which achieves faster processing speed (slightly more than 1 memory access per packet) and provides more accurate measurement results than the best existing work. Moreover, VHC performs well in a tight memory space (less than 1 bit per flow) where CT can no longer work. Extensive experiments based on real network trace data demonstrate the superior performance of VHC.

## VII. Acknowledgment

This work is supported in part by the National Science Foundation under grant CNS-00112539 and STC-1562485, and a grant from Florida Cybersecurity Center.

## References

[1] Cisco, "The Zettabyte Era-Trends and Analysis." [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/VNI_Hyperconnectivity_WP.html/

[2] A. Kumar, J. Xu, and J. Wang, "Space-code bloom filter for efficient per-flow traffic measurement," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 12, pp. 2327–2339, 2006.

[3] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter Braids: A Novel Counter Architecture for Per-Flow Measurement," *Proc. of ACM SIGMETRICS*, June 2008.

[4] Y. Lu and B. Prabhakar, "Robust Counting Via Counter Braids: An Error-Resilient Network Measurement Architecture," *Proc. of IEEE INFOCOM*, April 2009.

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

[6] M. Chen and S. Chen, "Counter Tree: A Scalable Counter Architecture for Per-Flow Traffic Measurement," *Proc. of IEEE ICNP*, November 2015.

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

[8] N. Duffield, C. Lund, and M. Thorup, "Learn More, Sample Less: Control of Volume and Variance in Network Measurement," *IEEE Transactions of Information Theory*, vol. 51, no. 5, pp. 1756–1775, 2005.

[9] X. Dimitropoulos, P. Hurley, and A. Kind, "Probabilistic Lossy Counting: An Efficient Algorithm for Finding Heavy Hitters," *Proc. of ACM SIGCOMM*, 2008.

[10] M. Yoon, T. Li, S. Chen, and J. Peir, "Fit a Spread Estimator in Small Memory," *Proc. of IEEE INFOCOM*, April 2009.

[11] M. Yu, L. Jose, and R. Miao, "Software defined traffic measurement with opensketch," *Proc. of NSDI*, pp. 29–42, 2013.

[12] Y. Zhou, S. Chen, Y. Zhou, M. Chen, and Q. Xiao, "Privacy-Preserving Multi-Point Traffic Volume Measurement Through Vehicle-to-Infrastructure Communications," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 12, pp. 5619–5630, 2015.

[13] Y. Zhou, Y. Zhou, S. Chen, and O. P. K, "Limiting Self-propagating Malware Based on Connection Failure Behavior," *Proc. of Seventh International Conference on Network and Communications Security (NCS)*, 2015.

[14] Y. Zhou, S. Chen, Z. Mo, and Q. Xiao, "Point-to-Point Traffic Volume Measurement through Variable-Length Bit Array Masking in Vehicular Cyber-Physical Systems," *Proc. of IEEE ICDCS*, pp. 51–60, 2015.

[15] Y. Zhou, Z. Mo, Q. Xiao, S. Chen, and Y. Yin, "Privacy-Preserving Transportation Traffic Measurement in Intelligent Cyber-physical Road Systems," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 5, pp. 3749–3759, 2016.

[16] Y. Zhou, Y. Zhou, S. Chen, and O. P. K, "Limiting Self-Propagating Malware Based on Connection Failure Behavior through Hyper-Compact Estimators," *arXiv preprint arXiv:1602.03153*, 2016.

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

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

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

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

[21] Q. Xiao, S. Chen, M. Chen, and Y. Ying, "Hyper-Compact Virtual Estimators for Big Network Data Based on Register Sharing," *Proc. of ACM SIGMETRICS*, pp. 417–428, 2015.