Fifty-second Annual Allerton Conference
Allerton House, UIUC, Illinois, USA
October 1 - 3, 2014

Highly Compact Virtual Maximum Likelihood Sketches for Counting Big Network Data

Zhen Mo Yan Qiao Shigang Chen
Department of Computer & Information Science & Engineering
University of Florida
Gainesville, FL, USA

Tao Li
College of Computer Science and Technology
University of Science and Technology of China
Hefei, Anhui, China

***Abstract*— As the Internet moves into the era of big network data, it presents both opportunities and technical challenges for traffic measurement functions, such as flow cardinality estimation, which is to estimate the number of distinct elements in each flow. Cardinality estimation has important applications in intrusion detection, resource management, billing and capacity planning, as well as big data analytics. Due to the practical need of processing network data in high volume and high speed, past research has strived to reduce the memory overhead for cardinality estimation on a large number of flows. One important thread of research in this area is based on sketches. The representative work includes the FM sketches [1], the LogLog sketches [2], and the HyperLogLog sketches [3]. Each sketch requires multiple bits and many sketches are needed for each flow, which results in significant memory overhead. This paper proposes a new method of virtual maximum likelihood sketches to reduce memory consumption. First, we design *virtual sketches* that use no more than two bits per sketch on average. Second, we design *virtual sketch vectors* that consider all flows together. Based on these new constructs, we design a flow cardinality solution with an online operation module and an offline estimation module. We also consider the problem of differentiated estimation that gives flows of high priorities better precision in their cardinality estimations. We implement the new solution and perform experiments to evaluate its performance based on real traffic traces.**

I. Introduction

As the Internet moves into the era of big network data, it presents both opportunities and technical challenges for data flow measurement at both the core and the edge of the networks. This paper focuses on a particular measurement function, counting *the number of distinct elements in each flow*, which is traditionally referred to as *flow cardinality* or *flow spread*. Flows and elements can be flexibly defined depending on application context. A few examples are given below.

- For scan detection, we can define each flow as all packets from the same source address and its elements as the destination addresses in the headers of the packets. The flow from a scanner has a large cardinality because it sends packets to many different destination addresses.
- For the gateway of a network to automatically identify its internal servers (possibly for resource alignment), it may regard all inbound packets to each internal address as a flow and the source addresses in the headers of the packets as elements. The flow to a server tends to have a larger cardinality than flows to other hosts because more clients communicate with the server. Moreover, as the gateway measures the cardinality of each flow periodically, it can detect potential DDoS attacks to a server if it finds the cardinality of the flow to the server jumps far beyond the normal value.
- In other applications, the flows do not have to be network traffic. For example, an online retailer may want to count the number of different customers that purchase one product after purchasing another one. These two products form a purchase association. We *logically* interpret all customers making the two purchases as a flow. Identifying purchase associations with large cardinalities help the retailer make effective followup suggestions to customers after they make their first purchases.

However, exact count for each flow will cause large memory and computation overhead. Because we count the number of *distinct* elements, in order to remove duplicates, the data structures may have to keep the elements that have been seen [4], which is costly. Note that counting the number of elements in each flow without removing duplicates is a related but different problem [5], [6], [7]. Fortunately, it is often not necessary for applications such as those listed above. As an example, for scan detection, suppose the cardinalities of the flows from normal sources are in tens or fewer. If the cardinality of the flow from a scanner is in thousands, even with some estimation error, we can still separate it from the normal ones based on the estimated cardinality. Various methods have been proposed for estimating the cardinalities of flows [8], [9], [10], [11], [12]. One important thread of research in this area is based on sketches. The representative work includes the FM sketches

978-1-4799-8009-3/14/$31.00 ©2014 IEEE

[1], the LogLog sketches [2], and the HyperLogLog sketches [3], which have been implemented in real systems. In a typical implementation, a LogLog or HyperLogLog sketch uses 5 bits, and an FM sketch uses more. Multiple sketches (in hundreds) are needed for each flow to ensure estimation accuracy.

There are practical needs for reducing per-flow memory overhead in cardinality estimation. If the cardinality estimation function is implemented on a network processor chip, because the on-chip memory is typically small and the number of flows in modern networks can be very large, we will have to minimize per-flow overhead in order to accommodate more flows. In the previous application example of purchase associations, if there are hundreds of thousands of different products, the number of possible purchase associations (flows) can in tens of billions. For such a large number of flows, memory overhead may become a problem.

This paper proposes a new method, called virtual maximum likelihood sketches, to reduce memory consumption by cardinality estimation on a large number of flows. It embodies two ideas. The first idea is called *virtual sketches*, which use no more than two bits per sketch on average, while retaining the functional equivalence to an FM sketch. The second idea is called *virtual sketch vectors*, which combine the sketches of all flows into a mixed common pool. Together, these two ideas can drastically reduce the overall memory overhead. Based on virtual sketches and virtual vectors, we design a cardinality estimation solution with an online operation module and an offline estimation module. For a system where some flows are more important than others, we investigate the problem of differentiating estimation accuracy, depending on the priorities of the flows. We implement the new solution and perform experiments based on real traffic traces. The results demonstrate that the new solution can work reasonably well in very tight space and has the ability of differentiating flows of different priorities.

The rest of the paper is organized as follows: Section II reviews the FM sketches and its followups. Section III describes the construction of virtual sketches and virtual sketch vectors, which serve as the basis for our new cardinality estimator. Section IV presents the online operation module and the offline estimation module for per-flow cardinality estimation. Section V introduces the idea of differentiated estimation accuracy with a solution by allowing multiple virtual sketch vectors per flow. Section VI presents the experimental results for evaluation of the proposed solution based on real traffic traces. Section VII draws the conclusion.

II. FM Sketches, LogLog, and HyperLogLog

This work is motivated from the famous FM sketches and probabilistic counting with stochastic averaging [1], which are designed to estimate the number of distinct elements in a multiset (or a flow in the context of this paper). For each flow, the data structures consist of s FM sketches, denoted as $S[j]$, $0 \le j < s$, each of which is a bitmap of l bits, denoted as $S[j][i]$, $0 \le i < l$.

To record an element of the flow, we first perform a uniformly distributed hash function on the element to select one of the sketches. Without loss of generality, we denote the selected sketch as $S[j]$. We then perform a geometrically distributed hash function on the element such that the probability for the output to be i is $2^{-(i+1)}$, $0 \le i < l$. Let v be the output. We set the bit $S[j][v]$ to one. Duplicate elements will set the same bit and thus automatically removed since they have no impact on the values of the sketches.

After recording all elements of the flow, we can estimate the flow cardinality from the sketches as follows: Let k be the true cardinality of the flow, $\hat{k}$ be the estimated cardinality, and $f(S[j])$ be the number of leading ones in sketch $S([j])$. For example, if $S[j]$ is 1111010...0, then $f(S[j]) = 4$. A functional relation can be developed between k and the expected number of leading zeros in each sketch. Replacing the expected value with the measured average of $\frac{\sum_{j=0}^{s-1} f(S[j])}{s}$, the following estimation formula is derived in [1]:

$$\hat{k} = \frac{s 2^{\frac{\sum_{j=0}^{s-1} f(S[j])}{s}}}{\theta}, \tag{1}$$

where $\theta = 0.77$ when k is sufficiently large. To ensure estimation accuracy, hundreds of sketches are often needed.

Instead of using the number of leading ones, the LogLog and HyperLogLog sketches [2], [3] develop their estimation formulas based on the highest index of any bit that is set to one in each sketch. Only $\log_2 l$ bits per sketch is needed to keep track of this index value. However, even though each sketch is smaller, hundreds of them are still needed to ensure accuracy.

This paper reduces per-flow memory usage in two ways. First, we develop virtual sketches, each of which uses no more than 2 bits on average. Second, we develop virtual sketch vectors, which are logically-separated but physically-shared data structures for a large number of different flows. Together, they can drive the memory usage down to the realm of one bit per flow.

III. Virtual Sketches

A. Virtual Sketches

The available physical memory B is divided into l bit arrays, denoted as $B[i]$, $0 \le i < l$, with the size of $B[i+1]$ being half of the size of $B[i]$. Let m be the number of bits in $B[0]$. The number of bits in $B[i]$ is $\frac{m}{2^i}$. The total number of bits in all bit arrays is $\sum_{i=0}^{l-1} m2^{-i} < 2m$.

The jth bit in $B[i]$ is denoted as $B[i][j]$, $0 \le j < \frac{m}{2^i}$. With each bit $B[0][j]$, we construct a *virtual sketch* $V[j]$ of l bits, denoted as $V[j][i]$, $0 \le i < l$, by taking one bit from every other array:

$$V[j][i] = B[i][j \bmod \frac{m}{2^i}]. \tag{2}$$

Because there are fewer bits in other arrays, their bits must be reused in multiple sketches. Figure 1 shows an example with $l = 3$ and $m = 8$. For instance, $V[0]$ consists of three

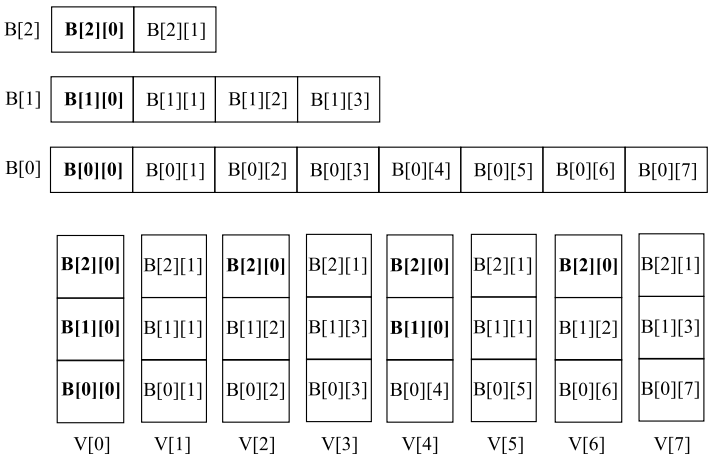

Fig. 1. An illustrative example of constructing virtual sketches from the bit arrays with $l = 3$ and $m = 8$. The first bit in each bit array is shown in bold text. To construct virtual sketches, the bits in the arrays except for $B[0]$ must be reused. The figure shows that the bits in $B[2]$ are each used four times in the virtual sketches, and the bits in $B[1]$ are each used twice.

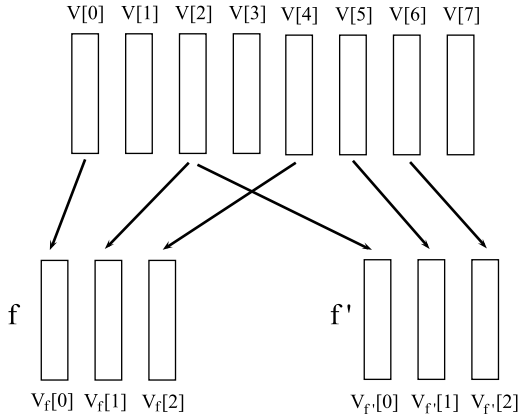

Fig. 2. An illustrative example of constructing virtual sketch vectors from the common pool V with $s = 3$. Consider two flows, f and f'. Three sketches are randomly drawn from V to form V_f. The same happens for $V_{f'}$. The virtual sketch $V[2]$ is used in both vectors.

bits, $B[0][0]$, $B[1][0]$ and $B[2][0]$, while $V[6]$ consists of three bits, $B[0][6]$, $B[1][2]$ and $B[2][0]$. They share $B[2][0]$.

In total, we construct m virtual sketches from fewer than $2m$ bits, using space fewer than 2 bits per sketch, more efficient than the existing sketches [1], [2], [3]. The m virtual sketches form a sketch pool, denoted as V.

B. Virtual Sketch Vector

For an arbitrary flow f, we select a number s of virtual sketches pseudo-randomly from the pool V to form a virtual sketch vector V_f for the flow, where s is a system parameter. For convenience, the sketches in the vector are also denoted as $V_f[j]$, $0 \leq j < s$, which will be used to record the elements in flow f.

There are different ways of selecting sketches from V to form V_f. One possible approach is described as follows: Let R be an array of s different constants that are randomly chosen. To select $V_f[j]$, we perform XOR on f and $R[j]$, and hash the result for an index to a sketch in V, where H is a hash function and f is the flow label. If the hash function requires a specific length of input and f has a different length, we can pad f or divide f into segments and XOR the segments such that the resulting length is appropriate. For simplicity, the formulas in the paper assume that the hash function can take input in the length of f. Hence,

$$V[H(f \oplus R[j])] \rightarrow V_f[j], 0 \leq j < s, \tag{3}$$

where $\oplus$ is the XOR operator and $\rightarrow$ means "*is selected for*." An example of constructing virtual sketch vectors for two flows is given in Figure 2.

In (3), $V[H(f \oplus R[j])]$ should be $V[H(f \oplus R[j]) \bmod m]$. We omit "mod m" to simplify the notation. We will use $V_f[j][i]$ for the ith bit of $V_f[j]$, $0 \leq i < l$.

In theory, we can construct an arbitrary number of virtual sketch vectors from the same pool V to support an arbitrary number of flows. The vectors for different flows will share sketches in V. Sharing causes noise. As the elements of flow f are recorded by the sketches in vector V_f, because those sketches are also used by the vectors of other flows, it introduces noise into other vectors. The more the number of flows is, the more the sharing of sketches will be, and the greater the noise will be. We will use the maximum likelihood method to remove the noise in each vector caused by other flows through sketch sharing.

IV. Counting Distinct Elements in Network Flows

A. Online Operation

Consider a device processing an incoming stream of data from a large number of flows. The device may be a router processing the arrival packets which belong to different flows. A measurement function implemented on the router can provide estimations of flow cardinalities during each measurement period. The device may also be a server processing sale records and estimate the number of occurrences for each purchase association. See the introduction for application examples.

When processing the incoming data stream, the device extracts a sequence of flow/element pairs. We introduce a *recording probability* β, i.e., each flow/element pair has a probability of β to be recorded. To implement the recording probability, each flow/element pair is first sampled with a probability of $\frac{\beta}{1-2^{-l}}$; its reason will become clear shortly. Consider an arbitrary flow/element pair which is sampled. Let f be the flow label and e be the element (which belongs to f). To record the element, the device does the following.

First, it pseudo-randomly selects a sketch from V_f. More specifically, it performs a hash $H(f \oplus e)$ in the range of $[0, s)$ and selects $V_f[H(f \oplus e)]$. If f and e have different lengths, we may pad e or divide e into segments and perform XOR on the segments such that its length is equal to that of f.

Second, the device chooses a bit from the selected sketch. To do so, it performs another hash $H'(f \oplus e)$. Let z be the number of leading zeros in $H'(f \oplus e)$. If $z \geq l$, e is not

recorded. If $z < l$, the bit $V_f[H(f \oplus e)][z]$ is chosen and set to one for recording e. That is,

$$V_f[H(f \oplus e)][z] := 1, \tag{4}$$

where $:=$ is the assignment operator. By (3), it becomes

$$V[H(f \oplus R[H(f \oplus e)])][z] := 1. \tag{5}$$

By (2), it becomes

$$B[z][H(f \oplus R[H(f \oplus e)]) \bmod \frac{m}{2^i}] := 1. \tag{6}$$

Clearly, multiple occurrences of the same flow/element pair will cause the same bit to be set, Therefore, duplicate elements in a flow are filtered out automatically. Only distinct elements may be recorded by different bits. It is also possible that two distinct elements set the same bit. Such collision is considered in our later development of estimation formula.

We stress that V and V_f are logical concepts that will help us derive a formula for estimating the flow cardinalities. They are not physically constructed during the phase of online operation. The only physical data structures that the device maintain are $B[i]$, $0 \le i < l$. The only operation per flow/element pair is sampling and then possibly assignment in (6). In the assignment, $\frac{m}{2^i}$ can be pre-computed. If m is chosen to be a power of 2, the modulo can be accomplished by a right-shift operation. The two hashes, $H(f \oplus e)$ and $H'(f \oplus e)$, can be combined into one: Suppose the hash output of $H(f \oplus e)$ has 32 bits, $s = 256$, and $l = 8$. The first eight bits of $H(f \oplus e)$ will be sufficient for selecting a sketch from V_f. The remaining 24 bits can substitute $H'(f \oplus e)$ for selecting a bit from the sketch; in fact, only 7 bits are needed since $l = 8$. In summary, the computation of (6) requires two hashes and one memory access, plus some simple operations such as XOR and shift.

The probability for $z = i$, $\forall i \ge 0$, is $2^{-(i+1)}$. The probability for $z \ge l$ is 2^{-l}. Hence, the probability for e to be recorded is the sampling probability $\frac{\beta}{1-2^{-l}}$ multiplied by $(1 - 2^{-l})$, which gives β.

The probability for e to set the ith bit of a particular sketch, denoted as p_i, is

$$p_i = \frac{\beta}{1-2^{-l}} \times \frac{1}{s} \times 2^{-(i+1)}, \tag{7}$$

where $0 \le i < l$. The reason is that e has a probability of $\frac{\beta}{1-2^{-l}}$ to be sampled, a probability of $\frac{1}{s}$ to select the sketch, and finally a probability of $2^{-(i+1)}$ to select the ith bit. The value of p_i decreases exponentially with respect to i.

B. Offline Estimation based on Maximum Likelihood Method

After processing an incoming data stream, the online device offloads the bit arrays to a server for long-term storage and offline query. It will then reset its arrays for the next data stream.

The server can estimate the number n_i of flow/element pairs recorded in B_i, $\forall i \in [0, l)$. This may be done through probabilistic counting [13]:

$$n_i \approx -\frac{m}{2^i} \ln V_i, \tag{8}$$

where V_i is the fraction of bits in B_i that are zeros.

Given a flow label f under query, the server estimates its cardinality based on the stored bit arrays B_i. In this offline operation, the server first constructs the virtual sketch vector V_f from the bit arrays. Combining (2) and (3), for $0 \le j < s$, $0 \le i < l$, we have

$$V_f[j][i] = B[i][H(f \oplus R[j]) \bmod \frac{m}{2^i}]. \tag{9}$$

Let k be the true cardinality of flow f. Consider an arbitrary bit $V_f[j][i]$. We derive the probability for the bit to be zero, which happens when (1) none of the k elements from f causes the bit to be set as one, and (2) none of the elements from other flows causes the bit to be set. Each element of f has a probability p_i in (7) to set $V_f[j][i]$ as one. The probability for none of the k elements from f to set it as one is $(1 - p_i)^k$.

$V_f[j][i]$ is a bit in $B[i]$. From (8), there are approximately n_i elements from all flows that set bits in $B[i]$. Among them, the number from flow f is approximately

$$k \times \frac{\beta}{1-2^{-l}} \times 2^{-(i+1)} = \frac{k\beta 2^{-(i+1)}}{1-2^{-l}}$$

because each element has a probability of $\frac{\beta}{1-2^{-l}}$ to be sampled and, regardless of which sketch is selected, it has a probability of $2^{-(i+1)}$ to set the ith bit in the sketch (which is a bit in $B[i]$). Hence, the number of elements from flows other than f, denoted as n'_i, is approximately

$$n'_i \approx n_i - \frac{k\beta 2^{-(i+1)}}{1-2^{-l}}. \tag{10}$$

For an arbitrary element from another flow, the chance for $V_f[j]$ to happen to be a sketch in the vector of that flow and be selected for recording the element is $\frac{s}{m} \times \frac{1}{s} = \frac{1}{m}$. Hence, the probability for none of the n'_i elements from other flows to set $V_f[j][i]$ is $(1 - \frac{1}{m})^{n'_i}$. Summarizing the above analysis, we have the following formula for the probability of $V_f[j][i]$ being zero:

$$\phi_i = (1 - p_i)^k (1 - \frac{1}{m})^{n'_i}. \tag{11}$$

Let $u_i = s - \sum_{j=0}^{s} V_f[j][i]$, which is the number of zeros at the ith bits of all sketches in V_f. The likelihood function of observing $\{u_i \mid 0 \le i < l\}$ with respect to k is

$$L(k) = \prod_{i=0}^{l-1} \binom{s}{u_i} \times \phi_i^{u_i} (1 - \phi_i)^{s-u_i}. \tag{12}$$

We find an estimated cardinality $\hat{k}$ for flow f that maximizes $L(k)$. That is,

$$\hat{k} = \max_{0 \le k \le K} \{L(k)\}, \tag{13}$$

where K is the maximum flow size of interest. We may solve (13) numerically through exhaustive search, which is however computationally costly. In our experiments, we adopt a bi-section search method, producing the same results as the exhaustive search: Denote the search range as $[r_1, r_2]$. Initially set $r_1 = 0$ and $r_2 = K$. Let $r_3 = \lfloor \frac{r1+r_2}{2} \rfloor$. Compute $L(r_3)$ and $L(r_3 + 1)$. If $L(r_3)$ is greater than $L(r_3 + 1)$, set

$r_2 = r_3$; if $L(r_3 + 1)$ is greater, set $r_1 = r_3$. Repeat the above procedure until $r_2 - r_1 \leq 1$. Finally, set $\hat{k}$ to be the one among r_1 and r_2 that produces a larger value of the likelihood function.

V. Differentiated Estimation Accuracy

Consider an online monitoring application where a gateway keeps track of the external scanning activities by measuring how many distinct destination addresses that each external source has contacted during each measurement period. All packets from the same source address constitute a flow, and the elements are destination addresses in the packet headers. The gateway may have access to a list of potentially malicious external hosts based on the past results from firewalls and other intrusion detection systems. Naturally, it is desirable to improve the accuracy in flow cardinality estimation for these flows over the background of other flows from addresses not in the list.

We introduce the problem of cardinality estimation with differentiated accuracy. Let $F = \{F_0, F_1, ..., F_{g-1}\}$ be the set of flows to be measured, which is divided into g subsets. Assume most flows belong to the base subset F_0. Their estimation accuracy serves as a baseline. Other subsets, F_v, $0 < v < g$, have increasingly higher requirements on estimation accuracy. Each F_v is assigned an integer *priority number* a_v such that $a_0 = 1$ and $a_v > a_{v-1}$. The differentiated accuracy requirement is that the variance of the cardinality estimation $\hat{k}$ for a flow f in F_v should be only $\frac{1}{a_v}$ of the variance if the same flow had belonged to the base subset F_0.

To meet the differentiated accuracy requirement, we propose to measure the cardinality of flow f independently for a_v times. That is, we assign a_v virtual sketch vectors to f, and every element from f is probabilistically recorded for a_v times, each time by a different vector. Each vector will give an estimation with a baseline accuracy comparable to similar flows in F_0, which have a single vector per flow. When we average the a_v estimations for flow f, the variance of the average is $\frac{1}{a_v}$ of the variance for each individual estimation. More details are given below.

We assume that the device performing online operation is able to classify the incoming sequence of flow/element pairs into the correct subsets such that each flow/element pair, f and e, is associated with a priority number a. The classification is beyond the scope of this paper. As an example, network traffic may be classified by pre-set ACLs or address lists.

If $a > 1$, we logically assign multiple virtual vectors to flow f, denoted as V_f^v, $0 \leq v < a$, which are constructed as follows:

$$V[H(f \oplus R[v \times a + j])] \rightarrow V_f^v[j], 0 \leq v < a, 0 \leq j < s, \tag{14}$$

where R is an array of $s \times a_{g-1}$ different constants. To record e, one bit from each V_f^v will be chosen and set to one. To do so, we need another array R' of a_{g-1} different constants. The two hash functions for selecting a bit in V_f^v are $H(f \oplus e \oplus R'[v])$ and $H'(f \oplus e \oplus R'[v])$. Following a process similar to that in Section IV-A, to record e, we can show that the device should set the following bits:

$$B[z_a][H(f \oplus R[H(f \oplus e \oplus R'[v])]) \bmod \frac{m}{2^i}] := 1,\ 0 \leq v < a, \tag{15}$$

where z_a is the number of leading zeros in $H'(f \oplus e \oplus R'[v])$.

During offline estimation, given a flow label f and its priority number a, we reconstruct the sketch vectors V_f^v, $0 \leq v < a$, from (14), compute an estimate $\hat{k}_v$ from each V_f^v, and return the average estimate:

$$\hat{k} = \frac{\sum_{v=0}^{a-1} \hat{k}_v}{a}. \tag{16}$$

VI. Experiments

A. Experiment Setup

We evaluate the proposed virtual maximum likelihood sketches (VMLS) through experiments based on real traffic traces. We obtained inbound packet header traces that were collected through Cisco's NetFlow from the main gateway at University of Florida for five days. The source address in the packet header is used as flow label, and the measurement period is one day. Hence, each flow consists of all packets from the same source address during a day. The destination address in the packet header is used as element. For each flow, its cardinality is the number of distinct destination addresses that the source has sent packets to. One application for such per-flow measurement is scan detection as explained in the introduction.

TABLE I
Traffic trace

	flows	flow/element pairs	avg cardinality
Day 1	3,558,510	10,048,129	2.82
Day 2	6,468,158	11,886,945	1.84
Day 3	5,189,371	11,858,928	2.29
Day 4	3,582,938	9,978,131	2.78
Day 5	4,007,256	10,702,677	2.67

Some statistics of the five-day traces are shown in Table I. We use one day's trace as an example: It has 5,189,371 distinct source IP addresses, meaning that there are 5,189,371 flows. It has 11,858,928 distinct source/destination pairs (flow/element pairs). Hence, the average cardinality per flow is 2.29.

We implement the online operation module and the offline estimation module of the proposed VMLS. The online operation uses a default memory space of one bit per flow, i.e., the total number of bits in all bit arrays $B[i]$, $0 \leq i < l$, is equal to the number of flows in the trace. The existing FM, LogLog, hyperLogLog sketches cannot work under such a tight memory space.

Unless specified otherwise, the parameters of VMLS are set as follows: $l = 4$, $s = 250$, and the sampling probability is 50%. Increasing l to a larger value will extend the estimation range without adding much space overhead due to the exponentially decreasing nature of additional bit arrays.

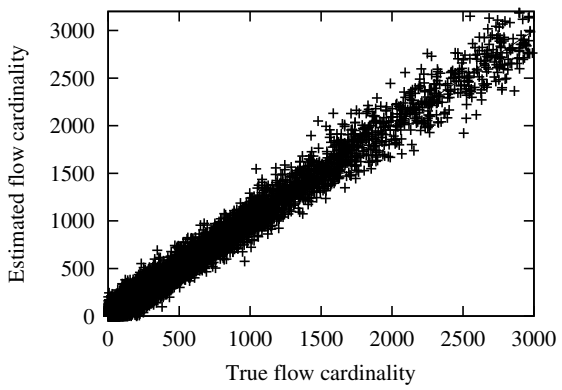

Fig. 3. Estimation accuracy of virtual maximum likelihood sketches with a single priority in memory of 1 bit per flow

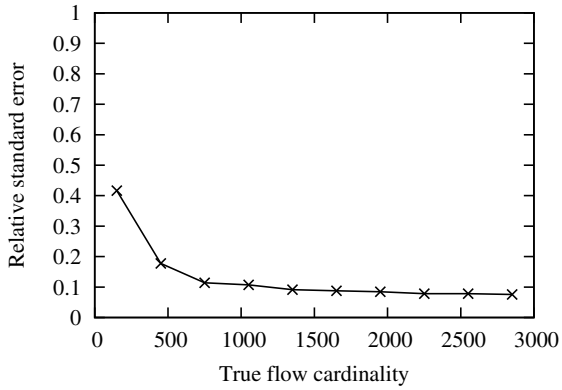

Fig. 4. Relative standard error of the estimations with a single priority in memory of 1 bit per flow

However, our traces do not have many large flows, and $l = 4$ is sufficient for the experiments.

B. Estimation Accuracy

The first experiment evaluates the estimation accuracy of VMLS with a single priority group, i.e., $g = 1$. For each one-day trace, we first apply the online operation module to record the elements of the flows. We then use the offline estimation module to compute an estimated cardinality for each flow. We then plot the five-day results in Figure 3. Each point in the figure represents a flow. Its x coordinate is the flow's true cardinality, k. Its y coordinate is the flow's estimated cardinality, $\hat{k}$. The closer a point is to the equality line $y = x$, the better the estimation accuracy will be. From the figure, we can see that the points cluster around the equality line, indicating reasonably good estimation accuracy even under a tight space of one bit per flow.

Figure 4 shows the relative standard error of the estimations. Because there are too few flows for some cardinality values, we compute the relative standard error by dividing the horizontal axis into measurement bins. In each bin, we compute the difference between the true cardinality and the estimated value for each flow, divide it by the true cardinality, square the result, add these squares for all flows, divide it by the number of flows minus one, and then take the squareroot. The figure shows that the relative standard error is large for small flows, but relatively small for large flows. It is below 10% for flows whose cardinalities are beyond 2,000. Even for small flows, although the relative standard error is large, the absolute error is still limited, which is evident from Figure 3, making it easy to separate large flows from small ones.

C. Differentiated Estimation Accuracy

The second experiment evaluates the differentiated estimation accuracy of VMLS with two priority groups, where $g = 2$, $a_0 = 1$ and $a_1 = 4$. 10% of all flows are randomly selected for the higher priority, and the remaining flows belong to the base priority. Figure 5 shows the estimation results of high-priority flows, and Figure 6 shows the results of base-priority flows. It can be seen that the estimation accuracy of the former is much better than that of the latter; the points representing high-priority flows are much closer to the equality line.

A more direct comparison is given in Figure 7, which shows that the standard errors for high-priority flows are about half of the errors for the base-priority flows when the flow cardinalities are beyond 500. This conforms to the accuracy requirement specified by $a_1 = 4$: the variance of high-priority flows is $\frac{1}{4}$ of the variance of base-priority flows. Hence, the standard deviation (or error) of the former should be half of the latter.

D. Varied Memory Availability

The third experiment evaluates the performance of VMLS with two priority groups under different memory availability. Figure 8-10 shows the results when the memory of the online operation module is 0.5 bit per flow. Comparing with Figure 5-7, the estimation errors are considerably larger, indicating that the practical value of VMLS will begin to diminish when the available memory is too small for the online module.

Figure 8-10 shows the results when the memory of the online operation module is 3 bits per flow. Comparing with Figure 5-7, the estimation errors are measurably better. For example, for flows of cardinalities around 2,000, the relative standard error of the base priority is 9.6% under memory of 1 bit per flow, and it is lowered to 6.3% under memory of 3 bits per flow. This result indicates that better estimation accuracy can be achieved through memory increase for the online module.

VII. Conclusion and Future Work

This paper proposes a new solution of virtual maximum likelihood sketches for cardinality estimation. It has four technical contributions: virtual sketches, virtual sketch vectors, a maximum likelihood method for cardinality estimation based on per-flow virtual sketch vectors, and a method to achieve differentiated estimation accuracy among multiple subsets of flows with different priorities. The experimental results based on real traffic traces demonstrate that the new

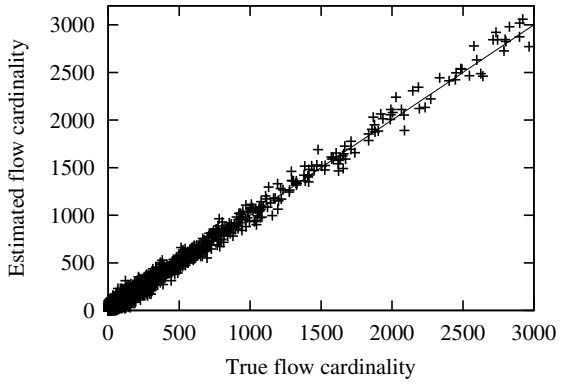

Fig. 5. Estimation accuracy of higher priority flows with $g_1 = 4$ in memory of 1 bit per flow

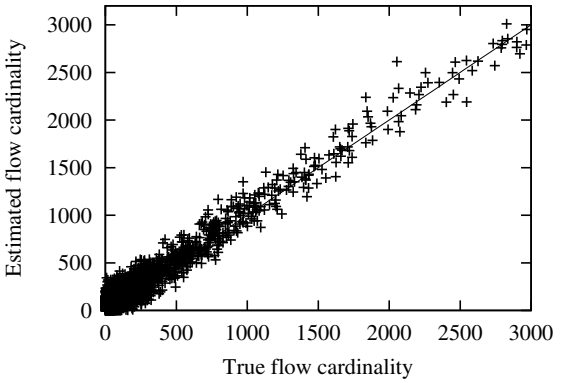

Fig. 8. Estimation accuracy of higher priority flows with $g_1 = 4$ in memory of 0.5 bit per flow

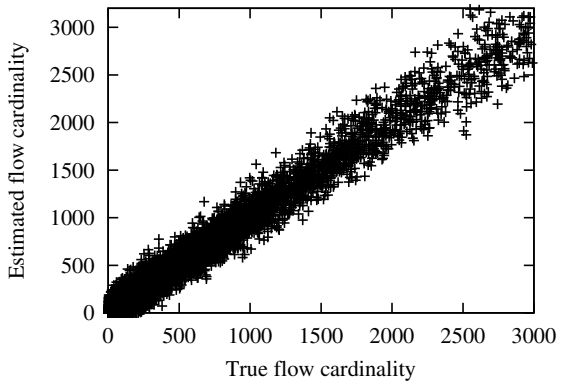

Fig. 6. Estimation accuracy of base priority flows with $g_0 = 1$ in memory of 1 bit per flow

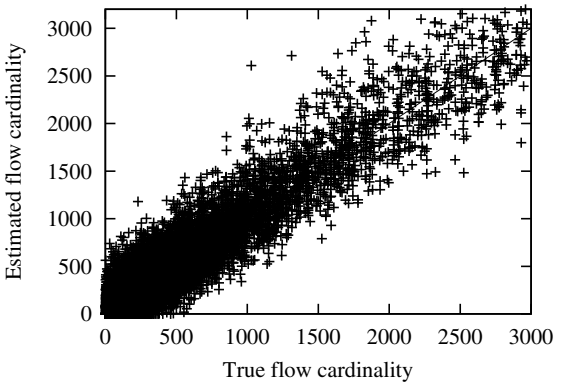

Fig. 9. Estimation accuracy of base priority flows with $g_0 = 1$ in memory of 0.5 bit per flow

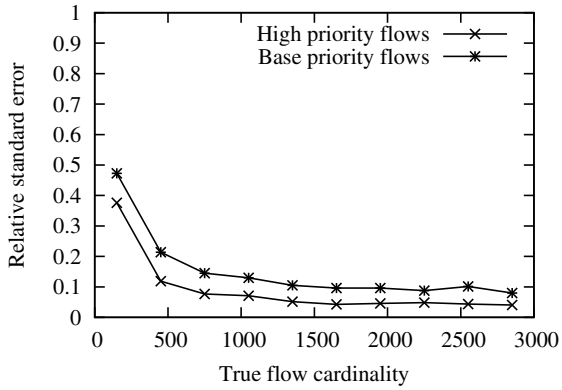

Fig. 7. Relative standard error of the estimations with two priorities in memory of 1 bit per flow

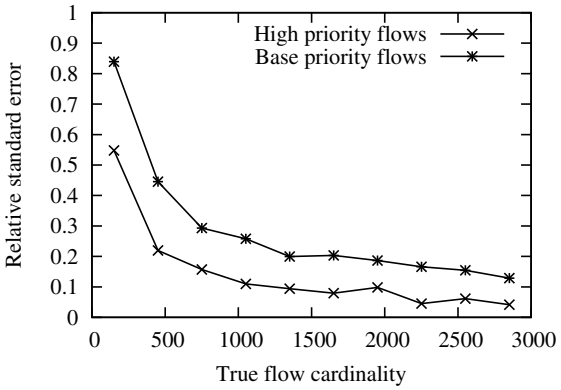

Fig. 10. Relative standard error of the estimations with two priorities in memory of 0.5 bits per flow

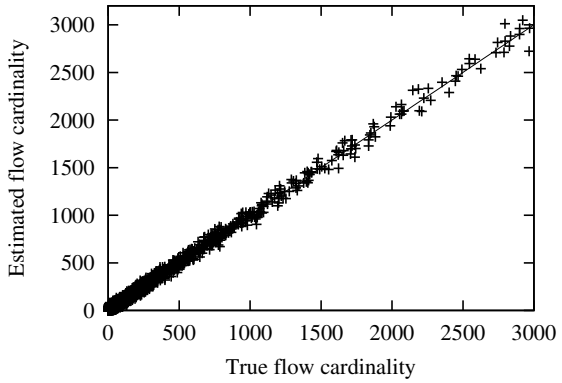

Fig. 11. Estimation accuracy of higher priority flows with $g_1 = 4$ in memory of 3 bits per flow

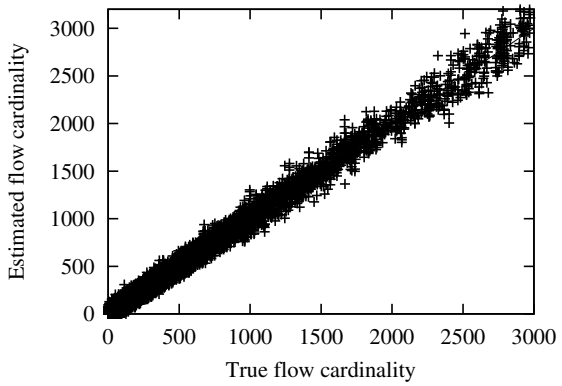

Fig. 12. Estimation accuracy of base priority flows with $g_0 = 1$ in memory of 3 bits per flow

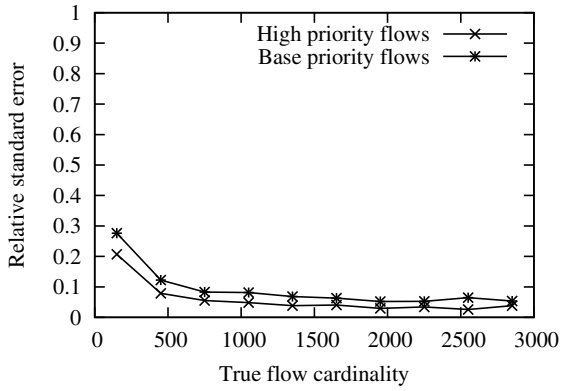

Fig. 13. Relative standard error of the estimations with two priorities in memory of 3 bits per flow

solution produces cardinality estimations with reasonable accuracy in very tight memory space.

Our future work is to formally analyze the performance of virtual maximum likelihood sketches, including the bias and the standard error of estimated cardinalities. We will study the impact of the system parameters, including s, β and l, on the performance, as well as how to set the optimal parameters. We will also find ways to reduce the relative standard errors of flows with small cardinalities.

VIII. Acknowledgement

This work was supported in part by the National Science Foundation under grant CNS-1115548 and CNS-1409797.

References

[1] P. Flajolet and G. N. Martin, "Probabilistic Counting Algorithms for Database Applications," *J. Comput. Syst. Sci.*, vol. 31, no. 2, 1985.

[2] M. Durand and P. Flajolet, "Loglog Counting of Large Cardinalities," *ESA: European Symposia on Algorithms*, pp. 605–617, 2003.

[3] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier., "HyperLogLog: The Analysis of a Near-optimal Cardinality Estimation Algorithm," *Proc. of AOFA: International Conference on Analysis Of Algorithms*, 2007.

[4] Q. Zhao, J. Xu, and A. Kumar, "Detection of Super Sources and Destinations in High-Speed Networks: Algorithms, Analysis and Evaluation," *IEEE JASC*, vol. 24, no. 10, pp. 1840–1852, 2006.

[5] 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.

[6] P. Lieven and B. Scheuermann, "High-Speed Per-Flow Traffic Measurement with Probabilistic Multiplicity Counting," *Proc. of IEEE INFOCOM*, pp. 1–9, 2010.

[7] T. Li, S. Chen, and Y. Ling, "Fast and Compact Per-Flow Traffic Measurement through Randomized Counter Sharing," *Accepted by IEEE INFOCOM*, 2011.

[8] Z. Bar-yossef, T. S. Jayram, R. Kumar, D. Sivakumar, L. Trevisan, and Luca, "Counting Distinct Elements in a Data Stream," *Proc. of RANDOM: Workshop on Randomization and Approximation*, 2002.

[9] A. Kumar, J. Xu, J. Wang, O. Spatschek, and L. Li, "Space-Code Bloom Filter for Efficient Per-Flow Traffic Measurement," *Proc. of IEEE INFOCOM*, March 2004, A journal version was published in IEEE JSAC, 24(12):2327-2339, December 2006.

[10] S. Venkatataman, D. Song, P. Gibbons, and A. Blum, "New Streaming Algorithms for Fast Detection of Superspreaders," *Proc. of NDSS*, 2005.

[11] C. Estan, G. Varghese, and M. Fish, "Bitmap Algorithms for Counting Active Flows on High-Speed Links," *IEEE/ACM Transactions on Networking (TON)*, vol. 14, no. 5, pp. 925–937, October 2006.

[12] M. Yoon, T. Li, S. Chen, and J.-K. Peir, "Fit a Compact Spread Estimator in Small High-Speed Memory," *IEEE/ACM Transactions on Networking*, vol. 19, no. 5, pp. 1253–1264, 2011.

[13] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor, "A Linear-time Probabilistic Counting Algorithm for Database Applications," *ACM Transactions on Database Systems*, vol. 15, no. 2, pp. 208–229, 1990.