This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2009 proceedings.

Fit a Spread Estimator in Small Memory

MyungKeun Yoon Tao Li Shigang Chen Jih-Kwon Peir
Department of Computer & Information Science & Engineering
University of Florida, Gainesville, FL 32611, USA
Email: {myoon, tali, sgchen, peir}@cise.ufl.edu

***Abstract*—The spread of a source host is the number of distinct destinations that it has sent packets to during a measurement period. A spread estimator is a software/hardware module on a router that inspects the arrival packets and estimates the spread of each source. It has important applications in detecting port scans and DDoS attacks, measuring the infection rate of a worm, assisting resource allocation in a server farm, determining popular web contents for caching, to name a few. The main technical challenge is to fit a spread estimator in a fast but small memory (such as SRAM) in order to operate it at the line speed in a high-speed network. In this paper, we design a new spread estimator that delivers good performance in tight memory space where all existing estimators no longer work. The new estimator not only achieves space compactness but operates more efficiently than the existing ones. Its accuracy and efficiency come from a new method for data storage, called virtual vectors, which allow us to measure and remove the errors in spread estimation. We perform experiments on real Internet traces to verify the effectiveness of the new estimator.**

I. Introduction

Traffic measurement in high-speed networks has many challenging problems [1], [2], [3], [4], [5]. In this paper, we study the problem of *spread estimation*, which is to estimate the number of distinct destinations to which each source has sent packets that are of all or certain specific types.

We define a *contact* as a source-destination pair, for which the source has sent a packet to the destination. In the most general terms, the source or destination can be an IP address, a port number, or a combination of them together with other fields in the packet header. The *spread* of a source is the number of distinct destinations contacted by the source during a measurement period. A *spread estimator* is a software/hardware module on a router (or firewall) that inspects the arrival packets and estimates the spread of each source. It must implement two functions. The first function is to store the contact information extracted from the arrival packets. The second function is to estimate the spread of each source based on the collected information. Although our discussion will focus on the source's spread, we may change the role of source and destination and use the same spread estimator to measure the *spread of a given destination*, which is the number of distinct sources that have contacted the destination.

A spread estimator has many important applications in practice. Intrusion detection systems can use it to detect port scans [6], in which an external host attempts to establish too many connections to different internal hosts or different ports of the same host. It can be used to detect DDoS attacks when too many hosts send traffic to a receiver [7], i.e., the spread of a destination is abnormally high. It can be used to estimate the infection rate of a worm by monitoring how many addresses the infected hosts will each contact over a period of time. A large server farm may use it to estimate the spread of each server (as a destination) in order to assess how popular the server's content is, which provides a guidance for resource allocation. An institutional gateway may use it to monitor outbound traffic and determine the spread of each external web server that has been accessed recently. This information can also be used as an indication of the server's popularity, which helps the local proxy to determine the cache priority of the web content.

The major technical challenge is how to fit a spread estimator in a small high-speed memory. Today's core routers forward most packets on the fast forwarding path between network interfaces that bypasses the CPU and main memory. To keep up with the line speed, it is desirable to operate the spread estimator in fast but expensive, size-limited SRAM [8]. Because many other essential routing/security/performance functions may also run from SRAM, it is expected that the amount of high-speed memory allocated for spread estimation will be small. Moreover, depending on the applications, the measurement period can be long, which requires the estimator to store an enormous number of contacts. For example, to measure the popularity of web servers, the measurement period is likely to be hours or even days. Hence, it is critical to design the estimator's data structure as compact as possible.

The past research meets the above challenge with several spread estimators [9], [10], [8] that process a large number of contacts in an ever smaller space. This paper adds a new member that not only requires far less memory than the best known one but also operates much more efficiently. It is able to provide good estimation accuracy in a tight space where all existing estimators fail. Our major contribution is a new methodology for contact storage and spread estimation based on *virtual vectors*, which use the available memory more efficiently for tracking the contacts of different sources.

Do we need a new spread estimator when there are already several? Consider the following scenario. Collected from the main gateway at the University of Florida on a day in 2005, the Internet traffic trace that we used in our experiments has around 10 million distinct contacts from 3.5 million distinct external sources to internal destinations. We expect that today's traffic on the Internet core routers will exceed these figures by far. Now assume the router can only allocate 1MB SRAM for the spread estimator. On average there are only 2.3 bits allocated for tracking the contacts from each source. We classify the existing estimators into several categories based

978-1-4244-3513-5/09/$25.00 ©2009 IEEE

on how they store the contact information: 1) storing per-flow information, such as Snort [11] and FlowScan [7], 2) storing per-source information, such as Bitmap Algorithms [9] and One-level/Two-level Algorithms[1] [10], and 3) mapping sources to the columns of a bit matrix, where each column stores contacts from all sources that are mapped to it, such as On-line Streaming Module (OSM) [8]. Obviously the first two categories will not work here because 2.3 bits are not enough to store the contacts of a source. As we will discuss shortly, OSM is also ineffective in this scenario because mapping multiple sources to one column introduces significant unremovable errors in spread estimation. Our new estimator uses a simple one-dimension bit array. A virtual bit vector is constructed from the array for each source. The virtual vectors share bits uniformly at random and introduce uniform errors in spread estimation that can be measured and removed. Based on virtual vectors, a spread estimator is mathematically developed and analyzed. The new estimator requires much less computation and fewer memory accesses than OSM, yet it can work in a very small memory where OSM cannot. Its performance is evaluated by experiments using real Internet traffic traces.

II. Existing Spread Estimators

Snort [11] maintains a record for each active connection and a connection counter for each source IP. Keeping per-flow state is too memory-intensive for a high-speed router, particularly when the fast memory allocated to the function of spread estimation is small.

One-Level/Two-Level Algorithms [10] maintain two hash tables. One stores all distinct contacts occurred during the measurement period, including the source and destination addresses of each contact. The other hash table stores the source addresses and a contact counter for each source address. A probabilistic sampling technique is used to reduce the number of contacts to be stored. However, instead of storing the actual source/destination addresses in each sampled contact, one can use bitmaps [9] to save space. Each source is assigned a bitmap where a bit is set for each destination that the source contacts. One can estimate the number of contacts stored in a bitmap based on the number of bits set [9]. An index structure is needed to map a source to its bitmap. It is typically a hash table where each entry stores a source address and a pointer to the corresponding bitmap. However, such a spread estimator cannot fit in a tight space where only a few bits are available for each source — not enough for a bitmap to work appropriately.

If we make each bitmap sufficiently long, we will have to reduce the number of them and there will not be enough bitmaps for all sources. One solution is to share each bitmap among multiple sources. Consider a simple spread estimator that uses a bit matrix whose *columns* are *bitmaps*. Sources are assigned to columns through a hash function. For each contact, the source address is used to locate the column and, through another hash function, the destination address is used to determine a bit in the column to be set. One can estimate the number of contacts stored in a column based on the number of bits set. However, the estimation is for contacts made by *all sources that are assigned to the column*, not for the contacts of a specific source under query.

The information stored for one source in a column is the *noise* for others that are assigned to the same column. One must remove the noise in order to estimate the spread correctly. To solve this problem, OSM (Online Streaming Module) [8] assigns each source randomly to l (typically three) columns through l hash functions, and it sets one bit in each column when storing a contact. A source will share each of its columns with a different set of other sources. Consequently the noise (i.e., the bits set by other sources) in each column will be different. Based on such difference, a method was developed to remove the noise and estimate the spread of the source [8].

OSM also has problems. Not only it increases the overhead by performing $l + 1$ hash operations, making l memory accesses and using l bits for storing each contact, but the noise can be too much to be removed in a *compact* memory space where a significant fraction of all bits (e.g., above 50%) are set. The columns that high-spread sources are assigned to have mostly ones; they are called *dense columns*, which present a high level of noise for other sources.[2] The columns that *only* low-spread sources are assigned to are likely to have mostly zeros; they are called *sparse columns*. As we observe in the experiments, in a tight space, dense columns will account for a significant fraction of all columns. The probability for a low-spread source to be assigned to l dense columns is not negligible. Since these dense columns will have many bits set at common positions, the difference-based noise removal will not work, and hence the spread estimation will be wrong. We will confirm the above analysis by the experimental results in Section V.

Also related is the detection of stealthy spreaders using online outdegree histograms in [12], which detects the event of collaborative address scan by a large number of sources, each scanning at a low rate. It is able to estimate the number of participating sources and the average scanning rate, but it cannot perform the task of estimating the spread of each individual source in the arrival packets.

III. Design of Compact Spread Estimator (CSE)

We first motivate the concept of virtual vectors that are used to store the contact information. We then design our compact spread estimator (CSE). Finally we discuss how to store source addresses.

A. Motivation for Virtual Vectors

Existing estimators divide the space into bitmaps and then allocate the bitmaps to sources. If we use per-source bitmaps and each bitmap has a sufficient number of bits, then the total

[1] A probabilistic sampling technique is used in [10] to reduce the number of contacts that are input to the estimator (at the expense of estimation accuracy). Naturally, it also reduces the number of sources appearing in input contacts. This technique can be equally applied to other estimators such as those in [9] and the one to be proposed in this paper. When we say per-source state, we refer to sources that appear in the contacts after sampling (if the sampling technique is used).

[2] Note that each high-spread source produces l dense columns.

memory requirement will be too big. If we share bitmaps, it is hard to remove the noise caused by sources that are assigned to the same bitmap. Resolving this dilemma requires us to look at space allocation from a new angle.

Our solution is to create a *virtual bit vector* for each source by taking bits uniformly at random from a common pool of available bits. In the previous estimators, *two bitmaps do not share any bit*. Two sources either do not cause noise to each other, or cause severe noise when they share a common bitmap — they share all bits in the bitmap. Each source experiences a different level of noise that cannot be predicted. In our estimator, two virtual vectors may share one or more (which is very unlikely) common bits. While each source has its own virtual vector to store its contacts, noise still occurs through the common bit between two vectors. However, there is a very nice property: Because the bits in virtual vectors are randomly picked, there is an equal probability for any two bits from different vectors to be the same physical bit. The probability for the contacts of one source to cause noise to any other source is the same. When there are a large number of sources, the noise that they cause to each other will be roughly uniform. Such uniform noise can be measured and removed. This property enables us to design a spread estimator for a tight space where the previous estimators will fail. The new estimator is not only far more accurate in spread estimation but also much more efficient in its online operations.

B. CSE: Storing Contacts in Virtual Vectors

Our compact spread estimator (CSE) consists of two components: one for storing contacts in virtual vectors, and the other for estimating the spread of a source. The first component will be described below, and the second will be in the next subsection.

CSE uses a bit array B of size m, which is initialized to zeros at the beginning of each measurement period. The ith bit in the array is denoted as $B[i]$. We define a *virtual vector* $X(src)$ of size s for each source address src, where $s \ll m$. It consists of s bits pseudo-randomly selected from B.

$$X(src) = (B[H_0(src)], B[H_1(src)], ..., B[H_{s-1}(src)]), \quad (1)$$

where H_i, $0 \le i \le s-1$, are different hash functions whose range is $[0..m-1]$. They can be generated from a single master hash function H_M.

$$H_i(src) = H_M(src \oplus R[i]) \quad (2)$$

where R is an array of s different random numbers and $\oplus$ is the XOR operator.

When a contact (src, dst) is received, CSE sets one bit in B and the location of the bit is determined by both src and dst. More specifically, the source address src is used to identify a virtual vector $X(src)$, and the destination address dst is used to determine a bit location i^* in the virtual vector.

$$i^* = H_M(dst) \mod s \quad (3)$$

From (2) and (3), we know that the i^*th bit in vector $X(src)$ is at the following physical location in B:

$$H_{i^*}(src) = H_M(src \oplus R[i^*]) = H_M(src \oplus R[H_M(dst) \bmod s]).$$

Hence, to store the contact (src, dst), CSE performs the following assignment:

$$B[H_M(src \oplus R[H_M(dst) \bmod s])] := 1. \quad (4)$$

We stress that setting one bit by (4) is the only thing that CSE does when storing a contact. It takes two hash operations and one memory access. The source's virtual vector, as defined in (1), is never explicitly computed until the spread estimation is performed on an offline machine (to be described shortly). The bit, which is physically at location $H_M(src \oplus R[H_M(dst) \bmod s])$ in B, is logically considered as a bit at location $(H_M(dst) \bmod s)$ in the virtual vector $X(src)$. Note that duplicate contacts will be automatically filtered because they are setting the same bit and hence have no impact on the information stored in B. Multiple different contacts may set the same physical bit. This is embodied in the probabilistic analysis when we derive the spread estimation formula.

C. CSE: Spread Estimation

At the end of the measurement period, one may query for the spread of a source src, i.e., the number of distinct contacts that src makes in the period. **Let k be the actual spread of src. The formula that CSE uses to compute the estimated spread $\hat{k}$ of src is**

$$\hat{k} = s \cdot \ln(V_m) - s \cdot \ln(V_s) \quad (5)$$

where V_m is the fraction of bits in B whose values are zeros and V_s is the fraction of bits in $X(src)$ whose values are zeros. The value of V_m and V_s can be easily found by counting zeros in B and $X(src)$, respectively. The first item, $(-s \cdot \ln(V_m))$, captures the noise, which is uniformly distributed in B and thus does not change for different sources. The second item, $(-s \cdot \ln(V_s))$, is the estimated number of contacts that are stored in $X(src)$, including the contacts made by src and the noise.

We expect that queries are performed after B is copied from the router's high-speed memory to an offline computer in order to avoid interfering with the online operations. Below we will derive (5) mathematically. Its accuracy and variance will be analyzed in the next section.

Some additional notations are given as follows. Let n be the number of distinct contacts from all sources during the measurement period, U_m be the random variable for the number of '0' bits in B, and U_s be the random variable for the number of '0' bits in the virtual vector $X(src)$. Clearly, $V_m = \frac{U_m}{m}$ and $V_s = \frac{U_s}{s}$.

Let A_j be the event that the jth bit in $X(src)$ remains '0' at the end of the measurement period and 1_{A_j} be the corresponding indicator random variable. We first derive the probability for A_j to occur and the expected value of U_s. For an arbitrary bit in $X(src)$, each of the k contacts made by src has a probability of $\frac{1}{s}$ to set the bit as one, and each of the contacts made by other sources has a probability of $\frac{1}{m}$ to set it as one. All contacts are independent of each other when setting bits in B. Hence,

$$Prob\{A_j\} = (1 - \frac{1}{m})^{n-k}(1 - \frac{1}{s})^k, \quad \forall j \in [0..s-1]$$

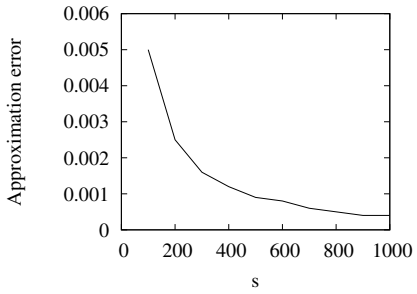

Fig. 1. The approximation error is very small when s is reasonably large.

Since U_s is the number of '0' bits in the virtual vector, $U_s = \sum_{j=0}^{s-1} 1_{A_j}$. Hence,

$$E(V_s) = \frac{1}{s}E(U_s) = \frac{1}{s}\sum_{j=0}^{s-1} E(1_{A_j}) = \frac{1}{s}\sum_{j=0}^{s-1} Prob\{A_j\}$$

$$= (1-\frac{1}{m})^{n-k}(1-\frac{1}{s})^k \quad (6)$$

$$\simeq e^{-\frac{n-k}{m}}e^{-\frac{k}{s}}, \ \ as \ \ (n-k), m, k, s \to \infty$$

$$\simeq e^{-\frac{n}{m}-\frac{k}{s}} \quad as \ \ k \ll m \quad (7)$$

The above equation can be rewritten as

$$k \simeq -s \cdot \frac{n}{m} - s \cdot \ln(E(V_s)). \quad (8)$$

Since the bits in any virtual vector are selected from B uniformly at random, the process of storing n contacts in the virtual vectors is to *set n bits randomly selected (with replacement) from a pool of m bits.* The mathematical relation between n and m has been given in [13] (in a database context) as follows.

$$n \simeq -m \cdot \ln(E(V_m)) \quad (9)$$

$$\text{where } E(V_m) = (1-\frac{1}{m})^n \quad (10)$$

Hence, (8) can be written as

$$k \simeq s \cdot \ln(E(V_m)) - s \cdot \ln(E(V_s)). \quad (11)$$

We have a few approximation steps above. In practice, n and m are likely to be very large numbers, the spread values (k) that are of interest are likely to be large, and s will be chosen large. The approximation errors that are accumulated in (11) can be measured as

$$\frac{|s \cdot \ln(E(V_m)) - s \cdot \ln(E(V_s)) - k|}{k} = |s \cdot \ln(\frac{1-\frac{1}{m}}{1-\frac{1}{s}}) - 1|$$

which is independent of n and k. This error is very small when s is reasonably large. For example, when $m = 1MB$, as shown in Fig. 1, the error is only 0.25% when s is 200.

Let $k_1 = -s \cdot \ln(E(V_m))$ and $k_2 = -s \cdot \ln(E(V_s))$. Eq (11) is rewritten as

$$k \simeq -k_1 + k_2.$$

Replacing $E(V_m)$ and $E(V_s)$ by the instance values, V_m and V_s, that are obtained from B and $X(src)$ respectively, we have the following estimation for k_1, k_2 and k.

$$\hat{k_1} = -s \cdot \ln(V_m) \quad (12)$$

$$\hat{k_2} = -s \cdot \ln(V_s) \quad (13)$$

$$\hat{k} = -\hat{k_1} + \hat{k_2} \quad (14)$$

According to Theorem A4 in [13], $\hat{k_1}$ is the maximum likelihood estimator (MLE) of k_1. Following a similar analysis, it is straightforward to see that $\hat{k_2}$ and $\hat{k}$ are the maximum likelihood estimators of k_2 and k, respectively. $\hat{k_1}$ is the noise, the estimated number of contacts made by others but inserted in $X(src)$ due to bit sharing between virtual vectors, and $\hat{k_2}$ estimates the total number of contacts stored in $X(src)$, including the noise.

When k is close to zero, it may happen with a small probability that V_s is greater than V_m due to statistical variance. In this case, we set $\hat{k} = 0$.[3]

D. System Architecture

The spread estimation system consists of a sampling module, CSE, and a module for Storing distinct Source Addresses, denoted as SSA. CSE has two sub-modules: one for Storing Contacts, denoted as CSE-SC, and the other for Spread Estimation, denoted as CSE-SE, which have been described in the previous two subsections. CSE-SC is located in the high-speed memory (such as SRAM) of a router, and CSE-SE is located on an offline computer answering spread queries.

The sampling module is used to handle the mismatch between the line speed and the processing speed of CSE-SC. In case that CSE-SC cannot keep up with the line speed, the source/destination addresses of each arriving packet will be hashed into a number in a range $[0, N)$. Only if the number is smaller than a threshold $T(< N)$, the contact is forwarded to CSE-SC. The threshold can be adjusted to match CSE-SC with the line speed. The final estimated spread of a source will become $\hat{k}\frac{N}{T}$. The focus of this paper is on CSE, assuming an incoming stream of contacts, regardless whether it comes from a sampling module or not.

Most applications, such as those we discuss in the introduction, are interested in high-spread sources. For them, we do not have to invoke SSA for each packet. When CSE-SC stores a contact at a bit in B, *only if the bit is set from '0' to '1'*, the source address is passed to the SSA module, which checks whether the address has already been stored and, if not, keeps the address. Comparing with CSE-SC, SSA operates infrequently. First, numerous packets may be sent from a source to a destination in a TCP/UDP session, but only the first packet may invoke SSA because the remaining packets will set the same bit. Second, while a source may send thousands or even millions of packets through a router, the number of times its address is passed to SSA will be bounded by s (which is the number of bits in the source's virtual vector). Hence, SSA can be implemented in the main memory, thanks to its infrequent operation.

For CSE-SE to work, m and s should be chosen large enough such that the noise introduced by other sources does not set all (or most) bits in a virtual vector. Hence, it is unlikely that the address of a high-spread source will not be stored in

[3]Such an estimation still provides useful information. It indicates that k is close to zero with high probability.

SSA. For example, even when only 10% of the bits in a virtual vector are not set by noise, for a source making 100 distinct contacts, the probability for none of its contacts being mapped to those 10% bits is merely $(1-10\%)^{100} = 2.65 \times 10^{-5}$.

IV. Analysis

We first study the mean and variance of $\hat{k_1}$ and $\hat{k_2}$, based on which we analyze the accuracy of the spread estimation $\hat{k}$.

A. Mean and Variance of $\hat{k_1}$ and $\hat{k_2}$

After setting n bits randomly selected from a pool of m bits, Whang [13] uses $\hat{n} = -m \ln V_m$ to estimate the value of n and gives the following results.

$$E(\hat{n}) = E(-m \ln V_m) \simeq n + \frac{e^{\frac{n}{m}} - \frac{n}{m} - 1}{2}$$
$$Var(\hat{n}) = Var(-m \ln V_m) \simeq m(e^{\frac{n}{m}} - \frac{n}{m} - 1)$$

Since $\hat{k_1} = -s \cdot \ln(V_m)$, we have

$$E(\hat{k_1}) \simeq \frac{s}{m}(n + \frac{e^{\frac{n}{m}} - \frac{n}{m} - 1}{2}) \quad (15)$$
$$Var(\hat{k_1}) \simeq \frac{s^2}{m}(e^{\frac{n}{m}} - \frac{n}{m} - 1). \quad (16)$$

If we choose an appropriate memory size m such that $m = O(n)$ and $e^{\frac{n}{m}} - \frac{n}{m} - 1$ is negligible when comparing with n, then $E(\hat{k_1}) \simeq s\frac{n}{m}$, which is indeed the average noise that a virtual vector of size s will receive when all n contacts are evenly distributed across the space of m bits. When m is large, the standard deviation, which is the square root of $Var(\hat{k_1})$, is insignificant when comparing with the mean.

Next we study $\hat{k_2}$. Let $\alpha = \frac{n}{m} + \frac{k}{s}$. Eq (7) can be rewritten as

$$E(V_s) \simeq e^{-\alpha}. \quad (17)$$

We derive $Var(V_s)$ in Appendix A, and it is

$$Var(V_s) \simeq \frac{1}{s}(e^{-\alpha} - e^{-2\alpha} - \frac{k}{s} \cdot e^{-2\alpha}). \quad (18)$$

In (13), $\hat{k_2}$ is a function of V_s. We expand the right-hand side of (13) by its Taylor series about $q = E(V_s) \simeq e^{-\alpha}$.

$$\hat{k_2}(V_s) = s \cdot (\alpha - \frac{V_s - q}{q} + \frac{(V_s - q)^2}{2q^2} - \frac{(V_s - q)^3}{3q^3} + ...) \quad (19)$$

Since $q = E(V_s)$, the mean of the second term in (19) is 0. Therefore, we keep the first three terms when computing the approximated value for $E(\hat{k_2})$.

$$E(\hat{k_2}) \simeq s \cdot (\alpha + \frac{1}{2q^2}E((V_s - q)^2))$$

$E((V_s - q)^2)) = Var(V_s)$ by definition. Applying (18), we have

$$E(\hat{k_2}) = s \cdot (\alpha + \frac{e^{\alpha} - 1 - \frac{k}{s}}{2s}) \quad (20)$$

If s is large enough such that $\frac{e^{\alpha} - 1 - \frac{k}{s}}{2s}$ is negligible, then $E(\hat{k_2}) \simeq s\alpha = s\frac{n}{m} + k$. Recall that $E(\hat{k_1}) \simeq s\frac{n}{m}$. Hence, $E(\hat{k}) = -E(\hat{k_1}) + E(\hat{k_2}) \simeq k$. In the next subsection, we will characterize more precisely the mean of $\hat{k}$ and how much it deviates from the true value of k.

TABLE I
Bias with respect to s and k

	k = 100	200	300	400	500	600	700	800
s= 400	0.54	0.77	1.05	1.47	2.04	2.82	3.85	5.21
s= 600	0.49	0.60	0.75	0.93	1.17	1.47	1.83	2.28
s= 800	0.47	0.54	0.63	0.75	0.88	1.05	1.24	1.47

To derive the variance of $\hat{k_2}$, we keep the first two items on the right-hand side of (19).

$$Var(\hat{k_2}) \simeq s^2 \cdot Var(\alpha - \frac{V_s - q}{q}) = \frac{s^2}{q^2} \cdot Var(V_s) \simeq s(e^{\alpha} - \frac{k}{s} - 1) \quad (21)$$

The combined impact of $V(\hat{k_1})$ and $V(\hat{k_2})$ on the variance of $\hat{k}$ will be studied next.

B. Estimation Bias and Standard Deviation

Based on the means of $\hat{k_1}$ and $\hat{k_2}$ derived previously, we obtain the mean of the spread estimation $\hat{k}$.

$$E(\hat{k}) = E(\hat{k_2}) - E(\hat{k_1}) \simeq s(\alpha + \frac{e^{\alpha} - 1 - \frac{k}{s}}{2s}) - \frac{s}{m}(n + \frac{e^{\frac{n}{m}} - \frac{n}{m} - 1}{2}) \quad (22)$$

The *estimation bias* is

$$E(\hat{k} - k) \simeq \frac{m(e^{\alpha} - 1 - \frac{k}{s}) - s(e^{\frac{n}{m}} - \frac{n}{m} - 1)}{2m} \quad (23)$$

As an example, for $n = 10,000,000$, $m = 2$ MB, and s = 400, 600 or 800, the bias with respect to k is shown in Table I. It is very small when comparing with the true spread k.

The variance of $\hat{k}$ is

$$Var(\hat{k}) = Var(\hat{k_1}) + Var(\hat{k_2}) - 2Cov(\hat{k_1}, \hat{k_2}) = Var(\hat{k_1}) + Var(\hat{k_2}) + 2\left[E(\hat{k_1})E(\hat{k_2}) - E(\hat{k_1}\hat{k_2})\right]. \quad (24)$$

We have already obtained $Var(\hat{k_1}), Var(\hat{k_2}), E(\hat{k_1})$ and $E(\hat{k_2})$, and thus only need to derive $E(\hat{k_1}\hat{k_2})$. Recall that $\hat{k_1} = s \cdot (-\ln(V_m))$ and $\hat{k_2} = s \cdot (-\ln(V_s))$. We expand $-\ln(V_m)$ and $-\ln(V_s)$ by their Taylor series about $p = e^{-\frac{n}{m}}$ and $q = e^{-\alpha}$, respectively.

$$\begin{aligned}
E(\hat{k_1}\hat{k_2}) &= s^2 E((-\ln(V_m))(-\ln(V_s))) \\
&= s^2 E((\frac{n}{m} - \frac{V_m - p}{p} + \frac{(V_m - p)^2}{2p^2} - ...) \\
&\quad \cdot (\alpha - \frac{V_s - q}{q} + \frac{(V_s - q)^2}{2q^2} - ...)) \\
&\simeq s^2\Big[\frac{n}{m}E(\alpha - \frac{V_s - q}{q} + \frac{(V_s - q)^2}{2q^2}) \\
&\quad + \alpha E(\frac{n}{m} - \frac{V_m - p}{p} + \frac{(V_m - p)^2}{2p^2}) - \frac{n}{m}\alpha\Big] \\
&= s^2\Big[\frac{n}{m}(\alpha + \frac{e^{\alpha} - 1 - \frac{k}{s}}{2s}) + \frac{\alpha}{m}(n + \frac{e^{\frac{n}{m}} - \frac{n}{m} - 1}{2}) - \frac{n}{m}\alpha\Big] \\
&= s^2\Big[\frac{n}{m}\alpha + \frac{\frac{n}{m}(e^{\alpha} - 1 - \frac{k}{s})}{2s} + \frac{\alpha(e^{\frac{n}{m}} - \frac{n}{m} - 1)}{2m}\Big] \quad (25)
\end{aligned}$$

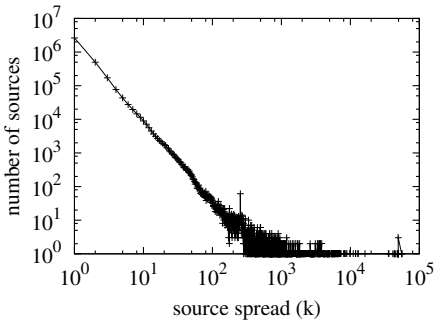

Fig. 2. Traffic distribution: each point shows the number of sources having a certain spread value.

From (15), (16), (20), (21), (24) and (25), we can obtain the closed-form approximation of $Var(\hat{k})$, which we omit. The standard deviation, divided by k to show the relative value, is

$$StdDev(\frac{\hat{k}}{k}) = \frac{\sqrt{Var(\hat{k})}}{k} \tag{26}$$

We have made a number of approximations, particularly, the truncation of less significant items in the Tayler series when deriving $Var(\hat{k_1}), Var(\hat{k_2}), E(\hat{k_1})$ and $E(\hat{k_2})$ and $E(\hat{k_1}\hat{k_2})$. The standard deviation embodies all those approximations. In Section V, Fig. 3-6, we will show the numerical values of the standard deviation calculated from (26) and compare them with the values measured from the experiments. The result demonstrates that the analytical approximations only introduce minor error when the source spread is not too small.

V. Experiments

We evaluate CSE through experiments using real Internet traffic traces. Our main goal in this paper is to provide a good spread estimator that can work in a small memory. In most of our experiments, the memory size, when averaging over all sources appearing in the input stream of contacts, ranges from 1.15 bits per source to 9.21 bits per source. Existing estimators that keep per-flow or per-source state [9], [10] will not work here as we have explained in Section II. The only related work that can still be implemented in such a small memory is OSM (Online Streaming Module) [8]; however, as the experimental results will demonstrate, it does not work well. Hence, CSE is valuable in the sense that it substantially extends the low end of memory requirement for the function of spread estimation in practice.

It should be noted that CSE makes two hash operations and one memory access for storing each contact, whereas OSM makes $l + 1$ hash operations and l memory accesses, where l is typically three. While CSE's efficient online operations are clearly advantageous for high-speed routers, our evaluation will focus on the area that is less quantified so far — the accuracy of spread estimation.

A. Experiment Setup

We obtained inbound packet header traces that were collected through Cisco's NetFlow from the main gateway at University of Florida for six days from April 1st to 6th, 2005. We implemented CSE and OSM, and executed them with the input of the six days' data. The experimental results are similar for those days. In this section, we will only present the results for the first day.

In our experiments, the source of a contact is the IP address of the packet sender, and the destination is the IP address of the receiver. The traffic trace on April 1 has 3,558,510 distinct source IP addresses, 56,234 distinct destination addresses, and 10,048,129 distinct contacts. The average spread per source is 2.84; namely, each source makes 2.84 distinct contacts on average. Fig. 2 shows the number of sources at each spread value in log scale. The number of sources decreases exponentially as the spread value increases from 1 to around 500. After that, there is zero, one or a few sources for each spread value.

We always allocate the same amount of memory to CSE and OSM for fair comparison. In each experiment, we feed the contacts extracted from the traffic trace to CSE or OSM, which stores the contact information in its data structure (located in SRAM or high-speed cache memory when deployed in a real router). The source addresses will be recorded in a separate data structure (located in the main memory because the operations for recording source addresses are performed infrequently as explained in Section III-D). After all contacts are processed, we use CSE or OSM to estimate the spread of each recorded source (which should be performed on an offline computer such as the network management center in practice).

B. Accuracy of Spread Estimation

The first set of experiments compare CSE and OSM in the accuracy of their spread estimations. CSE has two configurable parameters: the memory size m and the virtual vector size s. We perform four experiments with $m = 0.5$MB, 1MB, 2MB, and 4MB, respectively. In each experiment, we choose a value for s that minimizes the standard deviation as defined in (26) at $k = 250$, which is the middle point of the range (0..500) in which the spreads of most sources fall (Fig. 2).

OSM also has two configurable parameters: the memory size m and the column size (the number of rows in the bit matrix). The original paper does not provide a means to determine the best column size, but it suggests that 64 bits are typical. We tried many other sizes, and the performance of OSM under different column sizes will be presented shortly. After comparison, we choose the column size to be 128, which we believe is better than or comparable with other sizes for our experiments.

Fig. 3-6 present the experimental results when the memory allocated is 0.5MB, 1MB, 2MB and 4MB, respectively. Each figure has four plots from left to right. Each point in the first plot (CSE) or the second plot (OSM) represents a source, whose x coordinate is the true spread k and y coordinate is the estimated spread $\hat{k}$. The line of $\hat{k} = k$ is also shown. The closer a point is to the line, the more accurate the spread estimation is. To make the figure legible, when there are too many sources having a certain spread k, we randomly pick five to show in the first two plots. The third and fourth plots

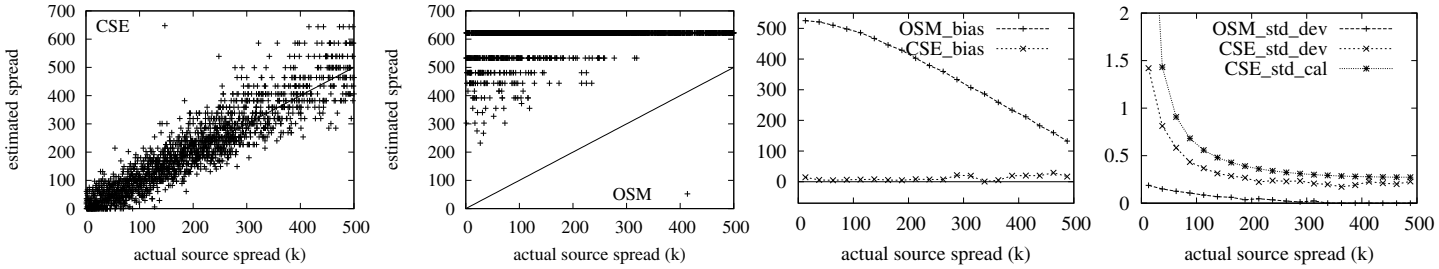

Fig. 3. $m = 0.5$MB. Each point in the first plot (CSE) or the second plot (OSM) represents a source, whose x coordinate is the true spread k and y coordinate is the estimated spread $\hat{k}$. The third plot shows the bias of CSE and OSM, which is the measured $E(\hat{k} - k)$ with respect to k. The fourth plot shows the standard deviation, which is the *measured* $\frac{\sqrt{Var(\hat{k})}}{k}$ for CSE and OSM, together with the *numerically-calculated* standard deviation for CSE based on (26) and (24).

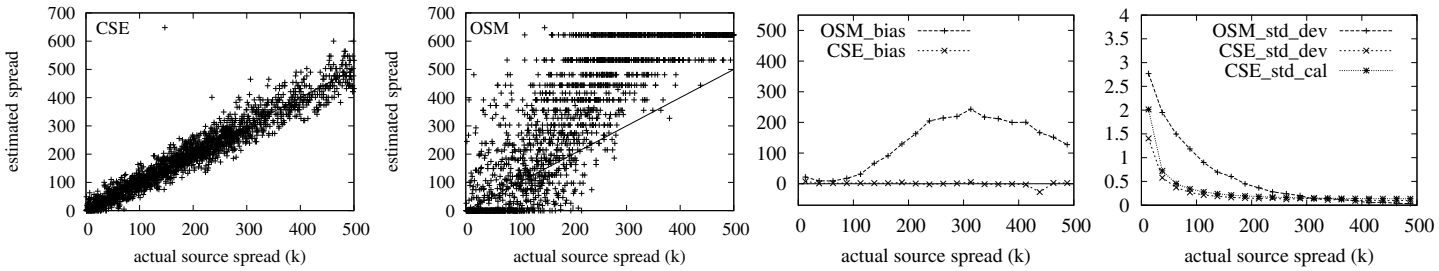

Fig. 4. $m = 1$M. See the caption of Fig. 3 for explanation.

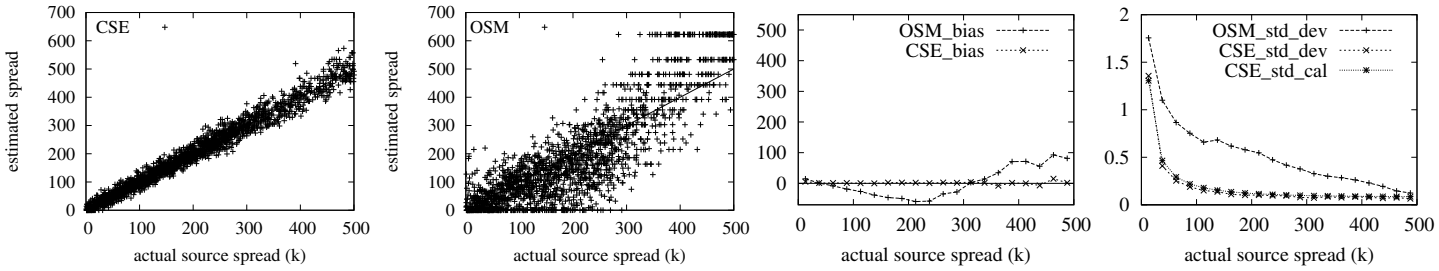

Fig. 5. $m =$2MB. See the caption of Fig. 3 for explanation.

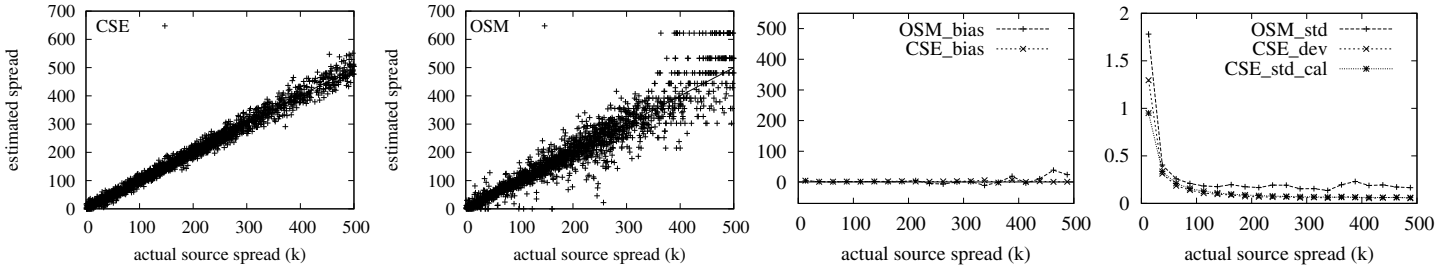

Fig. 6. $m =$4M. See the caption of Fig. 3 for explanation.

present the bias, $E(\hat{k}-k)$, and the standard deviation, $\frac{Var(\hat{k})}{k}$, measured in the experiment, respectively. Because there are too few sources for some spread values in our Internet trace, we divide the horizontal axis into measurement bins of width 25, and measure the bias and standard deviation in each bin. To verify the analytical result in Section IV, we also show the standard deviation numerically calculated from (26) and (24) as the curve under title "CSE_std_cal" in the fourth plot. We have the following experimental results.

- *First and Second Plots:* CSE works far better than OSM when the allocated memory is small. As the memory size increases, the performance of OSM improves and approaches toward the performance of CSE.
- *Third and Fourth Plots:* Both the bias and the standard deviation of CSE are much smaller than those of OSM. Moreover, the third plot shows that OSM is no longer a non-bias estimator when the memory is small. In fact, if we compare the absolute error $|\hat{k}-k|$ (that is not shown in the figures), the maximum absolute errors of CSE over the measurement bins are smaller than the average absolute errors of OSM in all four experiments.
- *Fourth Plot:* For CSE, the numerically-calculated standard deviation, which is the curve titled "CSE_std_cal", matches well with the experimentally-measured value, which is the curve titled "CSE_std_dev". It shows that the approximations made in the analysis do not introduce significant error.

C. Impact of Different s Values on Performance of CSE

The second set of experiments study the impact of different virtual-vector sizes s on the performance of CSE. We let m = 1MB and vary the value of s from 200 to 500,[4] while keeping the other parameters the same as in the previous set of experiments. Fig. 7 presents the bias and the standard deviation of CSE. The experimental results show that the performance of CSE is not very sensitive to the choice of s. A wide range of s gives comparable results. In the right plot of the figure, a larger s value leads to a slightly greater standard deviation for sources whose spreads (k) are small and a slightly smaller standard deviation for sources whose spreads are large. when k is larger.

D. Impact of Different Column Sizes on Performance of OSM

The third set of experiments demonstrate the impact of different column sizes on the performance of OSM. We let m = 1MB and vary the column size r from 64 to 512, while keeping the other parameters the same as in the first set of experiments. Fig. 8 presents the bias and the standard deviation of OSM. None of the r values makes OSM a non-bias estimator. When r is too large (such as 512), both bias and standard deviation are large. When r is too small (such as 64), its estimated spread does not go beyond 267, as shown in the left plot of Fig. 9. Comparing $r = 256$ and $r = 128$, the former leads to a much larger standard deviation, as shown in the right plot of Fig. 8. The impact of larger deviation can also be seen by comparing the right plot of Fig. 9 where $r = 256$ and the second plot in Fig. 4 where $r = 128$.

[4]In the experiment of Fig. 5, the s value, which minimizes the standard deviation at $k = 250$ as calculated from (26), is 286.

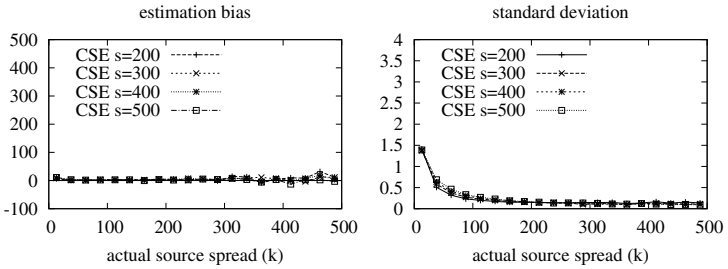

Fig. 7. The left plot shows the bias of CSE, which is the measured $E(\hat{k}-k)$ with respect to k. The right plot shows the standard deviation of CSE, which is the *measured* $\frac{\sqrt{Var(\hat{k})}}{k}$.

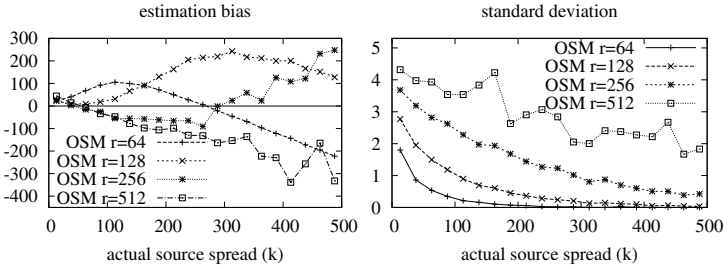

Fig. 8. The left plot shows the bias of OSM, which is the measured $E(\hat{k}-k)$ with respect to k. The right plot shows the standard deviation of OSM, which is the *measured* $\frac{\sqrt{Var(\hat{k})}}{k}$.

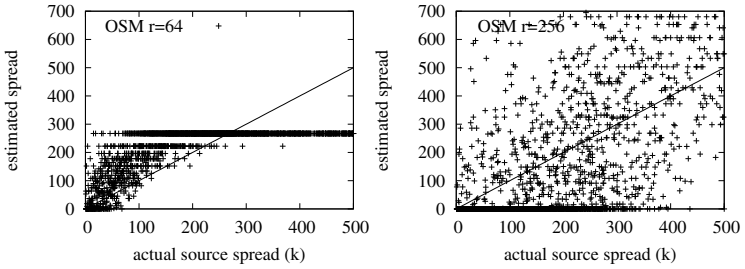

Fig. 9. The left plot shows the distribution of $(k, \hat{k})$ for all sources under OSM when $r = 64$, where k and $\hat{k}$ are the true spread and the estimated spread, respectively. The right plot shows the the distribution of $(k, \hat{k})$ OSM when $r = 256$.

E. An Application: Detecting Address Scan

Our last set of experiments compare CSE and OSM using an application for address scan detection. Suppose the security policy is to report all external sources that contact 250 or more internal destination during a day. If a source with a spread less than 250 is reported, it is called a *false positive*. If a source with a spread 250 or above is not reported, it is called a *false negative*. The *false positive ratio* (FPR) is defined as the number of false positives divided by the total number of sources reported. The *false negative ratio* (FNR) is defined as the number of false negatives divided by the number of sources whose spreads are 250 or more. The experimental results are shown in Table II. Clearly, CSE outperforms OSM by a wide

TABLE II
FALSE POSITIVE RATIO AND FALSE NEGATIVE RATIO WITH RESPECT TO MEMORY SIZE.

m(MB)	OSM		CSE	
	FPR	FNR	FPR	FNR
0.5	0.662	0.000	0.164	0.123
1	0.424	0.008	0.097	0.094
2	0.116	0.236	0.073	0.056
4	0.108	0.115	0.053	0.062

TABLE III
$\varepsilon = 10\%$, FALSE POSITIVE RATIO AND FALSE NEGATIVE RATIO WITH RESPECT TO MEMORY SIZE.

m(MB)	OSM		CSE	
	FPR	FNR	FPR	FNR
0.5	0.532	0.000	0.077	0.057
1	0.251	0.006	0.031	0.027
2	0.041	0.193	0.005	0.014
4	0.023	0.064	0.001	0.002

TABLE IV
$\varepsilon = 20\%$, FALSE POSITIVE RATIO AND FALSE NEGATIVE RATIO WITH RESPECT TO MEMORY SIZE.

m(MB)	OSM		CSE	
	FPR	FNR	FPR	FNR
0.5	0.401	0.000	0.023	0.022
1	0.135	0.002	0.001	0.006
2	0.013	0.146	0.000	0.002
4	0.006	0.030	0.000	0.000

margin when we take both FPR and FNR into consideration. The FNR is zero for OSM when $m = 0.5$MB. That is because OSM is a bias estimator in such a small memory. Its FPR is 66.2%

CSE also has non-negligible FPR and FNR because its estimated spread is not exactly the true spread. To accommodate impreciseness to a certain degree, the security policy may be relaxed to report all sources whose estimated spreads are $250 \times (1 - \varepsilon)$ or above, where $0 \leq \varepsilon < 1$. If a source whose true spread is less than $250 \times (1-2\varepsilon)$ gets reported, it is called an *ε-false positive*. If a source with a true spread 250 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 $\varepsilon = 10\%$ are shown in Table III, and those for $\varepsilon = 20\%$ are shown in Table IV, where the FPR and FNR for CSE are merely 0.1% and 0.6% respectively when $m = 1$MB.

VI. CONCLUSIONS

This paper proposes a new spread estimator that is able to provide good accuracy in a small memory where all existing estimators fail. It not only achieves space compactness but also operates more efficiently than the existing work. Our main technical contributions include a novel data structure based on virtual vectors, its operation protocol, and the corresponding formula for spread estimation, which is statistically analyzed and experimentally verified.

REFERENCES

[1] C. Estan and G. Varghese, "New Directions in Traffic Measurement and Accounting," *Proc. of ACM SIGCOMM'02*, October 2002.

[2] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-Based Change Detection: Methods, Evaluation, and Applications," *Proc. of IMC'03*, pp. 234–247, 2003.

[3] A. Kumar, M. Sung, J. Xu, and J. Wang, "Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Size Distribution," *Proc. of ACM SIGMETRICS*, 2004.

[4] 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.

[5] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lundn, "Online Identification of Hierarchical Heavy Hitters: Algorithms, Evaluation, and Application," *Proc. of ACM SIGCOMM IMC*, October 2004.

[6] S. Staniford, J. Hoagland, and J. McAlerney, "Practical Automated Detection of Stealthy Portscans," *Journal of Computer Security*, vol. 10, pp. 105 – 136, 2002.

[7] D. Plonka, "FlowScan: A Network Traffic Flow Reporting and Visualization Tool," *Proc. of USENIX LISA*, 2000.

[8] Q. Zhao, J. Xu, and A. Kumar, "Detection of Super Sources and Destinations in High-Speed Networks: Algorithms, Analysis and Evaluation," *IEEE JSAC*, vol. 24, no. 10, October 2006.

[9] C. Estan, G. Varghese, and M. Fish, "Bitmap Algorithms for Counting Active Flows on High-Speed Links," *IEEE/ACM Trans. on Networking*, vol. 14, no. 5, October 2006.

[10] S. Venkatataman, D. Song, P. Gibbons, and A. Blum, "New Streaming Algorithms for Fast Detection of Superspreaders," *Proc. of NDSS'05*, Feb. 2005.

[11] M. Roesch, "Snort–Lightweight Intrusion Detection for Networks," *Proc. of 13th Systems Administration Conference, USENIX*, 1999.

[12] Y. Gao, Y. Zhao, R. Schweller, S. Venkataraman, Y. Chen, D. Song, and M. Kao, "Detecting Stealthy Spreaders Using Online Outdegree Histograms," *Proc. of IEEE International Workshop on Quality of Service'07*, pp. 145–153, June 2007.

[13] K. Whang, B. Vander-Zanden, and H. Taylor, "A Linear Time Probabilistic Counting Algorithm for Database Applications," *ACM Transactions on Database Systems*, June 1990.

APPENDIX A: VARIANCE OF V_s

We derive the variance of V_s. The probability for A_i and A_j, $\forall i, j \in [0..s-1], i \neq j$, to happen simultaneously is

$$Prob\{A_i \cap A_j\} = (1 - \frac{2}{m})^{n-k}(1 - \frac{2}{s})^k.$$

Since $V_s = \frac{U_s}{s}$ and $U_s = \sum_{j=1}^{s} 1_{A_j}$, we have

$$\begin{aligned}
E(V_s^2) &= \frac{1}{s^2} E((\sum_{j=1}^{s} 1_{A_j})^2) \\
&= \frac{1}{s^2} E(\sum_{j=1}^{s} 1_{A_j}^2) + \frac{2}{s^2} E(\sum_{1 \leq i < j \leq s} 1_{A_j} 1_{A_j}) \\
&= \frac{1}{s}(1 - \frac{1}{m})^{n-k}(1 - \frac{1}{s})^k + \frac{s-1}{s}(1 - \frac{2}{m})^{n-k}(1 - \frac{2}{s})^k.
\end{aligned}$$

Based on (6) and the equation above, we have

$$\begin{aligned}
Var(V_s) &= E(V_s^2) - E(V_s)^2 \\
&= \frac{1}{s}(1 - \frac{1}{m})^{n-k}(1 - \frac{1}{s})^k + \frac{s-1}{s}(1 - \frac{2}{m})^{n-k}(1 - \frac{2}{s})^k \\
&\quad - (1 - \frac{1}{m})^{2(n-k)}(1 - \frac{1}{s})^{2k} \\
&= \frac{1}{s}((1 - \frac{1}{m})^{n-k}(1 - \frac{1}{s})^k - (1 - \frac{2}{m})^{n-k}(1 - \frac{2}{s})^k) \\
&\quad + (1 - \frac{2}{m})^{n-k}(1 - \frac{2}{s})^k - (1 - \frac{1}{m})^{2(n-k)}(1 - \frac{1}{s})^{2k} \\
&\simeq \frac{1}{s}(e^{-\alpha} - e^{-2\alpha}) + e^{-2\frac{n-k}{m} - 2\frac{k}{s}}(\frac{-k}{s^2}) \\
&\simeq \frac{1}{s}(e^{-\alpha} - e^{-2\alpha} - \frac{k}{s}e^{-2\alpha}). \qquad (27)
\end{aligned}$$