Spreader Classification Based on Optimal Dynamic Bit Sharing

Tao Li, Shigang Chen, *Senior Member, IEEE*, Wen Luo, Ming Zhang, *Student Member, IEEE*, and Yan Qiao

***Abstract*—Spreader classification is an online traffic measurement function that has many important applications. In order to keep up with ever-higher line speed, the recent research trend is to implement such functions in fast but small on-die SRAM. However, the mismatch between the huge amount of Internet traffic to be monitored and limited on-die memory space presents a significant technical challenge. In this paper, we propose an Efficient Spreader Classification (ESC) scheme based on *dynamic bit sharing*, a compact information storage method. We design a maximum likelihood estimation method to extract per-source information from the compact storage and determine the heavy spreaders. Our new scheme ensures that false positive/negative ratios are bounded. Moreover, given an arbitrary set of bounds, we develop a systematic approach to determine the optimal system parameters that minimize the amount of memory needed to meet the bounds. Experiments based on a real Internet traffic trace demonstrate that the proposed spreader classification scheme reduces memory consumption by 3–20 times when compared to the best existing work. We also investigate a new multi-objective spreader classification problem and extend our classification scheme to solve it.**

***Index Terms*—SRAM, Spreader classification, traffic measurement.**

I. Introduction

MODERN high-speed routers forward packets from incoming ports to outgoing ports via switching fabric, bypassing main memory and CPU. New technologies are pushing line speeds beyond OC-768 (40 Gb/s) to reach 100 Gb/s or even terabits per second [1]. The line cards in core routers must therefore forward packets at a rate exceeding 150 Mpps [2]; that leaves the processing time of each packet to be extremely small. Parallel processing and pipeline are used to speed up packet switching to a few clock cycles per packet [3]. In order to keep up with such high throughput, online network functions for traffic measurement, packet scheduling, access control, and quality of service will also have to be implemented using on-chip cache memory and bypassing main memory and CPU almost entirely [2], [4], [5]. However, fitting these network functions in fast but small on-chip memory represents a major technical challenge today [3].

The commonly used cache memory on network processor chips is SRAM, typically a few megabytes. Further increasing on-chip memory to more than 10 MB is technically feasible, but it comes with a much higher price tag, and access time is longer. There is a huge incentive to keep on-chip memory small because smaller memory can be made faster and cheaper. Off-chip SRAM is larger. However, it is slower to access. Hence, on-chip memory remains the first choice for online network functions that are designed to match the line speeds.

On-chip memory is limited in size. To make the matter even more challenging, it may have to be shared by multiple routing/performance/measurement/security functions that are implemented on the same chip. When multiple network functions share the same memory, each of them can only use a fraction of the available space. Depending on their relative importance, some functions may be allocated tiny portions of the limited memory, whereas the amount of data they have to process and store can be extremely large in high-speed networks. The disparity in memory demand and supply requires us to implement online functions as compact as possible [6], [7]. Furthermore, when different functions share the same memory, they may have to take turns to access the memory, making memory access the performance bottleneck. Since most online functions require only simple computations that can be efficiently implemented in hardware, their throughput will be determined by the bottleneck in memory access. Hence, we must also minimize the number of memory accesses made by each function when it processes a packet.

A. Online Measurement Function: Spreader Classification

We define a *contact* as a source–destination pair, for which the source sends a packet to the destination. The source or destination can be an IP address, a port number, a combination of address/port together with other fields in the packet header, or even a filename or URL in the payload. The *spread of a source* is the number of *distinct destinations* contacted by the source during a measurement period. Similarly, we can define the *spread of a destination*, which is the number of *distinct sources* that have contacted the destination.

It is very costly to measure the spread of each source (or destination) precisely. When a router measures the spread of a source, it has to remember the destinations that the source has contacted so far. Future packets from the source to the same destinations do not increase the spread value. The spread is increased only when a packet is sent to a new destination. The problem is that it takes too much memory to store all destination addresses that every source has contacted.

To solve this problem, various techniques such as sampling [8], probabilistic counting [9], Bloom filters [5], and bitmaps [6], [10], [11] are used to reduce memory overhead at the expense of measurement accuracy. The rationale is that absolutely precise measurement of spread values may not be necessary for most applications. It is often practically sufficient to estimate spread values with a certain level of

Manuscript received May 16, 2011; revised June 21, 2012; accepted July 23, 2012; approved by IEEE/ACM Transactions on Networking Editor T. Wolf. Date of publication October 16, 2012; date of current version June 12, 2013. This work was supported in part by the US National Science Foundation under grant CNS-1115548.

The authors are with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32608 USA (e-mail: tali@cise.ufl.edu; sgchen@cise.ufl.edu; wluo@cise.ufl.edu; mzhang@cise.ufl.edu; yqiao@cise.ufl.edu).

Digital Object Identifier 10.1109/TNET.2012.2218255

accuracy. Moreover, many applications only require us to classify spreaders into categories such as: 1) *heavy spreaders*, i.e., sources (or destinations) whose spread values are large, and 2) *non-heavy spreaders*. This further lowers the accuracy requirement and allows additional room for memory saving. For example, in scan detection, we want to identify heavy spreaders (scanners) that have contacted a lot of destinations. In the previous server-farm example, we want to know the set of servers with large spread values. Even if we do not identify all such servers, it is very helpful in resource allocation if we can identify most of them.

This paper studies the *spreader classification* problem. *Single-objective spreader classification* is to identify the set of heavy spreaders. *Multi-objective spreader classification* places sources (or destinations) into more categories based on their spread values. We will formally define these problems in Sections II and VI, respectively. For space conservation, we do not require precise measurement of each source's spread. Hence, the classification may not be 100% accurate. However, as we have argued previously, the results are useful if we can identify most heavy spreaders with bounded false positives (in which non-heavy spreaders are mistakenly reported as heavy spreaders) and bounded false negatives (in which heavy spreaders are not reported). This is particularly true when the false positive/negative ratios can be set arbitrarily small in a controllable tradeoff between classification accuracy and space overhead.

Classifying spreaders and identifying heavy spreaders have many applications. Intrusion detection systems can use them to detect port scans [12], in which an external heavy spreader attempts to establish too many connections to different internal hosts or different ports of the same host. They may be used to detect distributed denial-of-service (DDoS) attacks when too many hosts send traffic to a receiver [13], i.e., the spread of a destination is abnormally high. They can be used to monitor worm activities by identifying unusually high outbound contact volumes from internal hosts during a worm outbreak. A large server farm may use the spread values of its servers to find how popular the servers' content is, which provides guidance for resource allocation towards frequently accessed content. An institutional gateway may monitor outbound traffic and identify external Web servers that have large spread values. This information helps the local proxy learn the popularity of servers and determine the cache priority of Web content.

B. Prior Art

A related but *different* problem is to measure flow sizes [14]. We can use counters [15]–[17] or counter braids [4], [17] to measure flow sizes. If a source sends 1000 packets to a destination, the corresponding counter is increased by 1000. However, we cannot use counters to measure the spread values. If a source sends 1000 packets to the same destination, its spread is one. To remove duplicate packets that are sent to the same destination, per-source bitmaps are often used for spread estimation [10]. Note that bitmaps may also be used to count the number of packets under certain circumstances when duplication does not occur [18].

Cisco's Netflow provides coarse information (due to its aggressive sampling) about flow sizes, but does not gives spread information. It keeps per-flow state and stores source/destination addresses. Hence, it uses too much space to be implemented in SRAM.

Another line of related work [19]–[26] is on *heavy-hitter detection*, which is different from *spreader classification*. The former is to identify sources that send a large number of packets, whereas the latter is to identify sources that contact a large number of distinct destinations.

Solutions [5], [6], [11] that are fully equipped to estimate the spread values of all sources with decent accuracy are an overkill for spreader classification. Classification does not require accurate estimation for sources whose spread values are very low (clearly non-heavy spreaders) or very high (clearly heavy spreaders). Hence, it should cost much less space, as our experiments will demonstrate. There is limited work specifically for spreader classification. They either rely on ternary content addressable memory (TCAM) [27] or incur significant space overhead [8]. See Section VII for details.

C. Our Contribution

We propose an Efficient Spreader Classification (ESC) scheme based on a new storage method, called *dynamic bit sharing*, which stores contact information of all sources in a compact format. The level of compactness is so deep that the total number of available bits is less than 1/20 of the number of sources in some of our experiment cases: On average, just one bit is available for every 20 sources. Yet, still we are able to make spreader classification with predictable accuracy. Our scheme employs a maximum likelihood estimation method to extract per-source information from the compact storage and identify the heavy spreaders. It ensures that false positive/negative ratios are bounded. Moreover, given an arbitrary set of false positive/negative bounds, we develop a systematic approach to determine the optimal system parameters, such that the amount of memory needed to satisfy the bounds is minimized. We perform experiments based on a real traffic trace and demonstrate that, using these optimal parameters, we can reduce the memory consumption by 3–20 times when compared to the best existing work.

The rest of this paper is organized as follows. Section II gives the problem definition for single-objective spreader classification. Section III describes our dynamic bit-sharing scheme. Section IV presents the analytical results for optimal parameters. Section V presents the experimental results. Section VI solves the multi-objective spreader classification problem. Section VII describes the related work. Section VIII draws the conclusion.

II. Problem Statement

How do we formally define the classification objective? A straightforward objective is to report all sources whose spread values exceed a certain threshold. However, unless we measure the spread of each source precisely, we cannot accurately classify sources based on the threshold. As we have explained in the Introduction, precise spread measurement is a costly operation, and most existing work resorts to spread estimation. Naturally, the follow-up question is how to define a classification objective that embodies a probabilistic performance bound.

We adopt the *probabilistic classification objective* from [8]. Let h and l be two positive integers, $h > l$. Let α and β be two probability values, $0 < \alpha < 1$ and $0 < \beta < 1$. In each

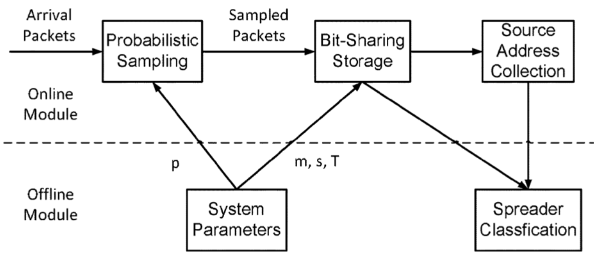


Fig. 1. Overall design.

measurement period, the objective is to report any source whose spread is h or larger with a probability no less than α and report any source whose spread is l or smaller with a probability no more than β. The measurement period may be defined as a fixed time interval or as a time interval during which a certain number of contacts is received. Let k be the spread of an arbitrary source src. The objective can be expressed in terms of conditional probabilities

$$Prob\{\text{report } src \text{ as a heavy spreader} \mid k \geq h\} \geq \alpha$$
$$Prob\{\text{report } src \text{ as a heavy spreader} \mid k \leq l\} \leq \beta. \quad (1)$$

We treat the report of a source whose spread is l or smaller as a *false positive*, and the nonreport of a source whose spread is h or larger as a *false negative*. Hence, the above objective can also be stated as bounding the false positive ratio by β and the false negative ratio by $1-\alpha$. Our goal is to minimize the amount of SRAM that is needed for achieving the above objective.

Although our technical discussion in this paper focuses on spreader classification of sources, the same techniques can be equally applied to spreader classification of destinations.

III. Efficient Spreader Classification Scheme

This section presents our ESC scheme, which is the combination of probabilistic sampling, dynamic bit sharing, and maximum likelihood estimation.

A. Overall Design

As shown in Fig. 1, our traffic measurement function consists of an online module and an offline module. The online module consists of three components for probabilistic sampling, bit-sharing storage, and source address collection. The probabilistic sampling component performs only one hash operation for each arrival packet. The bit-sharing storage component performs two hash operations and sets just one bit in a bitmap. It does so only for each sampled packet; it does nothing for packets that are not sampled. The source address collection component is involved less frequently—once for each TCP/UDP session; it just stores the source address. In this design, we try to keep the online module very simple and efficient.

The offline module consists of two components for computing the optimal system parameters and performing heavy spreader classification based on the data provided by the online module. While their computation is more complex, they are performed offline.

B. Probabilistic Sampling

To save space, a router samples the contacts made by external sources to internal destinations, and it only stores the sampled contacts. The router selects contacts for storage uniformly at random with a sampling probability p. The sampling procedure is simple: The router hashes the source/destination address pair of each packet that arrives at the external network interface into a number in a range $[0, N)$. If the hash result is smaller than $p \times N$, the contact will be stored; otherwise, the contact will not be stored.

C. Bit-Sharing Storage

For each source, a bit vector (also called *bitmap*) may be used to store all its sampled contacts. The bits are initially zeros. Each sampled contact is hashed to a bit in the bitmap, and the bit is set to one. At the end of the measurement period, using the method of probabilistic counting [9] or its variants [10], [11], we can estimate the number of contacts (i.e., the spread of the source) based on the number of zeros remaining in the bitmap.

However, using per-source bitmaps is not memory-efficient. If each bitmap is 32 bits long and there are 1M sources with sampled contacts, the total memory requirement will be 32 Mb. If the allocated SRAM space is much smaller, e.g., 0.5 Mb, there will not be enough bitmaps for all sources. This problem cannot be solved simply by reducing the size of each bitmap because even if a bitmap has just one bit, it still takes 1 Mb. Moreover, the performance of probabilistic counting [9] or its variants [10], [11] requires bitmaps to be not too small.

Our solution is to mix all bitmaps together and let them share bits, such that an almost arbitrary number of bitmaps can be created from a limited available space. Bit sharing among bitmaps causes information interference, which will be removed when we derive our formula. The level of bit sharing, which is determined by system parameters (see Section IV), controls the tradeoff between classification accuracy and space overhead. Details of our method are presented as follows.

Let m be the total number of available bits. All bits are organized in a single array B. For an arbitrary source src, we use a hash function to pseudorandomly select a number of bits from B to store the contacts made by src. The indices of the selected bits are $H(src \oplus R[0]), H(src \oplus R[1]), \ldots, H(src \oplus R[s-1])$, where $H(\ldots)$ is a hash function whose range is $[0, m)$, R is an integer array, storing randomly chosen constants whose purpose is to arbitrarily alter the hash result, and s ($\ll m$) is a system parameter that specifies the number of bits to be selected. The above bits form a *logical bitmap* of source src, denoted as $LB(src)$.

Similarly, a logical bitmap can be constructed from B for any other source. Essentially, we embed the bitmaps of all possible sources in B. The bit-sharing relationship is dynamically determined on the fly as new sources are allocated logical bitmaps from B.

At the beginning of a measurement period, all bits in B are reset to zeros. Consider an arbitrary contact $\langle src, dst \rangle$ that is sampled for storage, where src is the source address and dst is the destination address. The router sets *a single bit* in B to one. Obviously, it must also be a bit in the logical bitmap $LB(src)$. The index of the bit to be set for this contact is given as follows:

$$H(src \oplus R[H(dst \oplus K) \bmod s]).$$

The outer hash, $H(src \oplus R[\ldots])$, ensures that it is a bit in $LB(src)$. The inner hash, $H(dst \oplus K)$, ensures that the bit is pseudorandomly selected from $LB(src)$. The private ke K is introduced to prevent the *hash collision attacks*. In such an attack, a heavy spreader src finds a set of destination addresses, $dst_1, dst_2, \ldots$, that have the same hash value, $H(dst_1) = H(dst_2) = \ldots$ If it only contacts these destinations, the same bit in $LB(src)$ will be set, which allows the heavy spreader to stay undetected. This type of attack can be prevented if we use a cryptographic hash function such as MD5 or SHA1, which makes it difficult to find destination addresses that have the same hash value. However, if a weaker hash function is used for performance reasons, then a private key becomes necessary. Without knowing the key, the heavy spreaders will not be able to predict which destination addresses produce the same hash value.

To store a contact, ESC only sets a single bit. This is more efficient than other methods [8], [5] that require setting multiple bits for storing each contact.

D. Maximum Likelihood Estimation and Heavy Spreader Classification

At the end of the measurement period, ESC will send the content of B to an offline data processing center. There, the logical bitmap of each source src is extracted, and the estimated spread $\hat{k}$ of the source is computed. Only if $\hat{k}$ is greater than a threshold value T, ESC reports the source as a heavy spreader. We will discuss how to keep track of the source addresses in Section III-E, and explain how to determine the threshold T based on a given classification objective in Section IV. The formula for computing the estimated spread $\hat{k}$ is

$$\hat{k} = \frac{\ln V_s - \ln V_m}{\ln\left(1 - \frac{p}{s}\right) - \ln\left(1 - \frac{p}{m}\right)} \tag{2}$$

where p is the sampling probability, s is the size of the logical bitmap $LB(src)$, m is the size of B, V_s is the fraction of bits in $LB(src)$ whose values are zeros, and V_m is the fraction of bits in B whose values are zeros. We formally derive this formula as follows.

Let k be the true spread of source src, n be the number of distinct contacts made by all sources, and U_s be the number of bits in $LB(src)$ whose values are zeros. Clearly, $V_s = \frac{U_s}{s}$. Depending on the context, U_s (or V_s, V_m) is used either as a random variable or an instance value of the random variable.

The probability for any contact to be sampled for storage is p. Consider an arbitrary bit b in $LB(src)$. A sampled contact made by src has a probability of $\frac{1}{s}$ to set b to "1," and a sampled contact made by any other source has a probability of $\frac{1}{m}$ to set b to "1." Hence, the probability $q(k)$ for b to remain "0" at the end of the measurement period is

$$q(k) = \left(1 - \frac{p}{m}\right)^{n-k} \left(1 - \frac{p}{s}\right)^{k}. \tag{3}$$

Each bit in $LB(src)$ has a probability of $q(k)$ to remain "0." The observed number of "0" bits in $LB(src)$ is U_s. The likelihood function for this observation to occur is given as follows:

$$L = q(k)^{U_s}(1 - q(k))^{s-U_s}. \tag{4}$$

Following the standard process of maximum likelihood estimation, we derive the estimated spread $\hat{k}$ in (2). The details can be found in Appendix A.

We give an intuitive explanation for (2). First, we discuss how its development has taken information interference (due to bit sharing) into consideration.

- In (3), the term $(1 - \frac{p}{m})^{n-k}$ captures the effect of interference; it is the probability that a bit in $LB(src)$ is not set by any contact from a different source. Eventually, this results in two terms, $\ln V_m$ and $\ln(1 - \frac{p}{m})$, in (2).
- In (3), the term $(1 - \frac{p}{s})^{k}$ represents the actual effect of the source that we want to estimate. It is the probability that a bit in $LB(src)$ is not set by any contact from this source. Eventually, this results in two other terms, $\ln V_s$ and $\ln(1 - \frac{p}{s})$, in (2).

Second, the formula (2) is an estimation. It does not give the precise value of k due to its probabilistic bit setting nature. However, our goal is not precise measurement. We want to classify spreaders based on (1). Our analysis will demonstrate that, as long as the system parameters are set appropriately, the objective in (1) will be met.

E. Source Addresses

Collecting source addresses is treated as a separate task. If there is not enough SRAM to keep them, ESC stores source addressed in DRAM. It avoids storing the source address of every arrival packet. Instead, it stores a source address only when a contact sets a bit in B from "0" to "1." The frequency of storing source addresses is by far smaller than the packet arrival rate due to the following reasons. First, numerous packets may be sent from a source to a destination in a TCP/UDP session. Only the first sampled packet may cause the source address to be stored because only the first packet sets a bit from "0" to "1," and the remaining packets will set the same bit (which is already "1"). Therefore, for any TCP/UDP session, no matter how many packets it has, it triggers source address storage at most once. Second, a source may send thousands or even millions of packets through a router, but the number of times its address will be stored is bounded by s (which is the number of bits in the source's logical bitmap). Third, each arrival packet has a probability of p to be sampled. When it is not sampled, it has no chance to trigger source address storage. In summary, because the operation of storing source addresses is relatively infrequent, these addresses can be stored in the main memory.

IV. Optimal System Parameters and Minimum Memory Requirement

In this section, we first derive the probability for ESC to report an arbitrary source as a heavy spreader. Based on this probability, we develop the constraints that the system parameters must satisfy in order to achieve the classification objective in (1). Using these constraints, we determine the optimal values for the size s of the logical bitmaps, the sampling probability p, and the threshold T. We also determine the minimum amount of memory m that should be allocated for ESC to achieve the objective.

A. Report Probability

We first derive the probability for ESC to report an arbitrary source as a heavy spreader. Consider an arbitrary source src. ESC reports src as a heavy spreader if its estimated spread $\hat{k}$ exceeds a threshold T. The probability for this to happen, $\text{Prob}\{\hat{k} \geq T\}$, is given in (6), and it is derived as follows. From (2), we know that the following inequalities are equivalent:

$$\hat{k} \geq T$$
$$\frac{\ln V_s - \ln V_m}{\ln\left(1-\frac{p}{s}\right) - \ln\left(1-\frac{p}{m}\right)} \geq T$$
$$V_s \leq V_m \left(\frac{1-\frac{p}{s}}{1-\frac{p}{m}}\right)^T$$
$$U_s \leq s \cdot V_m \left(\frac{1-\frac{p}{s}}{1-\frac{p}{m}}\right)^T = C.$$

For a set of parameters m, s, p, and T, we define a constant

$$C = s \cdot V_m \cdot \left(\frac{1-\frac{p}{s}}{1-\frac{p}{m}}\right)^T$$

where the instance value of V_m can be measured from B after the measurement period. Hence, the probability for ESC to report src is $\text{Prob}\{\hat{k} \geq T\} = \text{Prob}\{U_s \leq C\}$.

U_s follows the binomial distribution with parameters s and $q(k)$, where $q(k)$ in (3) is the probability for an arbitrary bit in $LB(src)$ to remain zero at the end of the measurement period. Hence, the probability of having exactly i zeros in $LB(src)$ is given by the following probability mass function:

$$\text{Prob}\{U_s = i\} = \binom{s}{i} \cdot q(k)^i \cdot (1-q(k))^{s-i}. \quad (5)$$

We must have

$$\text{Prob}\{\hat{k} \geq T\} = \text{Prob}\{U_s \leq C\} = \sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(k)^i \cdot (1-q(k))^{s-i}. \quad (6)$$

After we obtain the probability of reporting an arbitrary source as a heavy spreader, we can derive the formulas for satisfying our system constraints in the following.

B. Constraints for the System Parameters

We now derive the constraints that the system parameters must satisfy in order to achieve the classification objective in (1). The result can be found in (9) and (10).

First, we give the variance of V_m, which is derived in Appendix B

$$\text{Var}(V_m) \simeq \frac{e^{-\frac{np}{m}}\left(1-\left(1+\frac{np^2}{m}\right)e^{-\frac{np}{m}}\right)}{m}. \quad (7)$$

It approaches to zero as m increases. In Fig. 2, we plot the ratio of the standard deviation $\text{Std}(V_m) = \sqrt{\text{Var}(V_m)}$ to $E(V_m)$, which can be found in (23). The figure shows that $\text{Std}(V_m)/E(V_m)$ is very small when m is reasonably large. In this case, we can approximately treat V_m as a constant

$$V_m \simeq E(V_m) \simeq \left(1-\frac{p}{m}\right)^n. \quad (8)$$

The classification objective can be stated as two requirements. First, the probability for ESC to report a source with $k \geq h$ must be at least α. That is, $\text{Prob}\{\hat{k} \geq T\} \geq \alpha, \forall k \geq h$. From (6), this requirement can be written as the following inequality:

$$\sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(k)^i \cdot (1-q(k))^{s-i} \geq \alpha$$

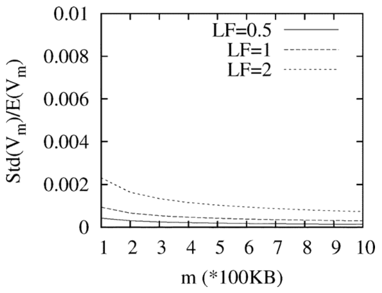


Fig. 2. Relative standard deviation, $\frac{\text{Std}(V_m)}{E(V_m)}$, approaches to zero as m increases. The *load factor* (LF) is defined as $n \cdot p/m$, where $n \cdot p$ is the number of distinct contacts that are sampled by ESC for storage. In our experiments (reported in Section V), when we use the system parameters determined by the algorithm proposed in this section, the load factor never exceeds 2.

where $C = s \cdot V_m \cdot (\frac{1-\frac{p}{s}}{1-\frac{p}{m}})^T \simeq s \cdot (1-\frac{p}{m})^n \cdot (\frac{1-\frac{p}{s}}{1-\frac{p}{m}})^T$. The left side of the inequality is an increasing function in k. Hence, to satisfy the requirement in the worst case when $k = h$, the following constraint for the system parameters must be met:

$$\sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(h)^i \cdot (1-q(h))^{s-i} \geq \alpha. \quad (9)$$

Second, the probability for ESC to report a source with $k \leq l$ must be no more than β. This requirement can be similarly converted into the following constraint:

$$\sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(l)^i \cdot (1-q(l))^{s-i} \leq \beta. \quad (10)$$

C. Optimal System Parameters

Our goal is to optimize the system parameters such that the memory requirement, m, is minimized under the constraints (9) and (10). The problem is formally defined as follows:

$$\begin{aligned} &\text{Minimize} && m \\ &\text{Subject to} && \sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(h)^i \cdot (1-q(h))^{s-i} \geq \alpha \\ &&& \sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(l)^i \cdot (1-q(l))^{s-i} \leq \beta \\ &&& C = s \cdot \left(1-\frac{p}{m}\right)^n \cdot \left(\frac{1-\frac{p}{s}}{1-\frac{p}{m}}\right)^T. \end{aligned} \quad (11)$$

The parameters, h, l, α, and β, are specified in the classification objective. There are two ways to define a measurement period (Section II). If the measurement period is defined as a fixed time interval, the value of n has to be estimated based on historical data [8]. To be conservative, we take the maximum number n^* of distinct contacts observed in a number of previous measurement periods. More specifically, (23) in Appendix A can be turned into a formula for estimating n in each previous period if we replace $E(V_m)$ with the instance value V_m

$$\hat{n} = -\frac{m}{p} \ln V_m. \quad (12)$$

We derive the relative bias and the relative standard deviation of the above estimation

$$\text{Bias}\left(\frac{\hat{n}}{n}\right) = E\left(\frac{\hat{n}}{n}\right) - 1 \simeq \frac{e^{\frac{np}{m}} - \frac{np^2}{m} - 1}{2np} \tag{13}$$

$$\text{Std}\left(\frac{\hat{n}}{n}\right) = \frac{\sqrt{m}}{np}\left(e^{\frac{np}{m}} - \frac{np^2}{m} - 1\right)^{1/2}. \tag{14}$$

They both approach to zero as m increases. Based on the largest $\hat{n}$ value observed in a certain number of past measurement periods, we can set the value of n^*. However, if the number of contacts in each period swings wildly, we should use the other definition of measurement period as a time interval during which a certain given number n of contacts are received; it is important to note that, in this definition, n is a "fixed" input parameter set by the network administrator.

A similar definition is adopted by [4] and [14], where a measurement period ends when a certain number of packets is received. In our case, after all system parameters such as m and p are computed (using the procedure to be described), during the runtime the router will be able to determine when each period ends based on a condition derived from (12): Replacing $\hat{n}$ with n, we have $V_m = e^{-\frac{np}{m}}$, and thus $U_m = m \times e^{-\frac{np}{m}}$. The right side can be precomputed as a threshold, and the value of U_m (i.e., the number of ones in B) can be kept in a register. When the value of the register reaches the threshold, the period ends.

To solve the constrained optimization problem (11), we need to determine the optimal values of the remaining three system parameters, s, p, and T, such that m will be minimized. We consider the left side of (9) as a function $F_h(m, s, p, T)$, and the left side of (10) as $F_l(m, s, p, T)$. Namely

$$F_h(m, s, p, T) = \sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(h)^i \cdot (1 - q(h))^{s-i}$$

$$F_l(m, s, p, T) = \sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(l)^i \cdot (1 - q(l))^{s-i}.$$

Both of them are *nonincreasing functions in* T, according to the relation between C and T. In the following, we present an iterative numerical algorithm to solve the optimization problem. The algorithm consists of four procedures.

• First, we construct a procedure called $\text{Potential}(m, s, p)$, which takes a value of m, a value of s and a value of p as input and returns the maximum value of $F_h(m, s, p, T)$ under the condition that $F_l(m, s, p, T) \le \beta$ is satisfied. Because $F_h(m, s, p, T)$ is a nonincreasing function in T, we need to find the smallest value of T that satisfies $F_l(m, s, p, T) \le \beta$. That can be done numerically through binary search: Pick a small integer T_1 such that $F_l(m, s, p, T_1) \ge \beta$ and a large integer T_2 such that $F_l(m, s, p, T_2) \le \beta$. We iteratively shrink the difference between them by resetting one of them to be the average $\frac{T_1+T_2}{2}$, while maintaining the inequalities, $F_l(m, s, p, T_1) \ge \beta$ and $F_l(m, s, p, T_2) \le \beta$. The process stops when $T_1 = T_2$, which is denoted as T^*. The procedure $\text{Potential}(m, s, p)$ returns $F_h(m, s, p, T^*)$. The pseudocode is presented in Algorithm 1 in Appendix C.

Essentially, what $\text{Potential}(m, s, p)$ returns is the maximum value of the left side in (9) under the condition that (10) is satisfied. The difference between $\text{Potential}(m, s, p)$ and α provides

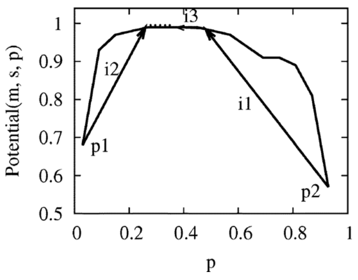


Fig. 3. (A) The curve (without the arrows) shows the value of $\text{Potential}(m, s, p)$ with respect to p when $m = 0.45$ MB and $s = 150$. Its nonsmooth appearance is due to $\lfloor C \rfloor$ in the formula of $F_h(m, s, p, T^*)$. $F_h(m, s, p, T^*)$ depends on the values of $\lfloor C \rfloor$ and $q(h)$, which are both functions of p. (B) The arrows illustrate the operation of $\text{OptimalP}(m, s)$. In the first iteration (arrow i_1), p_2 is set to be $(p_1 + p_2)/2$. In the second iteration (arrow i_2), p_1 is set to be $(p_1 + p_2)/2$. In the third iteration (arrow i_3), p_2 is set to be $(p_1 + p_2)/2$.

us with a quantitative indication on how conservative or aggressive we have chosen the value of m. If $\text{Potential}(m, s, p) - \alpha$ is positive, it means that the performance achieved by the current memory size is more than required. We shall reduce m. On the contrary, if $\text{Potential}(m, s, p) - \alpha$ is negative, we shall increase m.

Given the above semantics, when we determine the optimal values for p and s, our goal is certainly to maximize the return value of $\text{Potential}(m, s, p)$.

• Second, given a value of m and a value of s, we construct a procedure $\text{OptimalP}(m, s)$ that determines the optimal value p^* such that $\text{Potential}(m, s, p^*)$ is maximized. When the values of m and s are fixed, $\text{Potential}(m, s, p)$ becomes a function of p. It is a curve as illustrated in Fig. 3; see explanation under caption (A) and ignore the arrows in the figure for now.

We use a binary search algorithm to find a near-optimal value of p. Let $p_1 = 0$ and $p_2 = 1$. Let δ be a small positive value (such as 0.001). Repeat the following operation: Let $\bar{p} = (p_1 + p_2)/2$. If $\text{Potential}(m, s, \bar{p}) < \text{Potential}(m, s, \bar{p} + \delta)$, set p_1 to be $\bar{p}$; otherwise, set p_2 to be $\bar{p}$. The above iterative operation stops when $p_2 - p_1 < \delta$. The procedure $\text{OptimalP}(m, s)$ returns $(p_1 + p_2)/2$, which is within $\pm\delta/2$ of the optimal. This difference can be made arbitrarily small when we decrease δ at the expense of increased computation overhead. We want to stress that it is one-time overhead (not online overhead) to determine the system parameters before deployment. The operation of $\text{OptimalP}(m, s)$ is illustrated by the arrows in Fig. 3; see explanation under caption (B). The pseudocode is given in Algorithm 2 in Appendix C.

• Third, given a value of m, we construct a procedure $\text{OptimalS}(m)$ that determines the optimal value s^* such that $\text{Potential}(m, s^*, \text{OptimalP}(m, s^*))$ is maximized. When the value of m is fixed, $\text{Potential}(m, s, \text{OptimalP}(m, s))$ becomes a function of s. It is a curve as illustrated in Fig. 4. We can use a binary search algorithm similar to that of $\text{OptimalP}(m, s)$ to find s^*. The pseudocode is given in Algorithm 3 in Appendix C.

• Fourth, we construct a procedure $\text{OptimalM}()$ that determines the minimum memory requirement m^* through binary search: Denote $\text{Potential}(m, \text{OptimalS}(m), \text{OptimalP}(m, \text{OptimalS}(m)))$ as $\text{Potential}(m, \ldots)$. Pick a small value m_1 such that $\text{Potential}(m_1, \ldots) \le \alpha$, which means that the classification objective is not met—more specifically, according to the

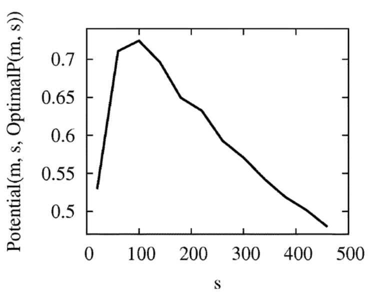


Fig. 4. Value of $\text{Potential}(m, s, \text{OptimalP}(m, s))$ with respect to s when $m = 0.25$ MB.

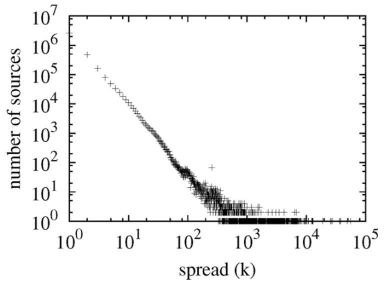


Fig. 5. Traffic distribution: Each point shows the number of sources having a certain spread value.

semantics of $\text{Potential}(\ldots)$, the constraint (9) cannot be satisfied if the constraint (10) is satisfied. Pick a large value m_2 such that $\text{Potential}(m_2, \ldots) \geq \alpha$, which means that the classification objective is met. Repeat the following operation. Let $\bar{m} = \lfloor (m_1 + m_2)/2 \rfloor$. If $\text{Potential}(\bar{m}, \ldots) \leq \alpha$, set m_1 to be $\bar{m}$; otherwise, set m_2 to be $\bar{m}$. The above iterative operation terminates when $m_1 = m_2$, which is returned by the procedure $\text{OptimalM}()$. The pseudocode is given in Algorithm 4 in Appendix C.

In practice, a network administrator will first define the classification objective that is specified by α, β, h, and l. Depending on the definition of measurement period, n is a given input or is set based on historic data. If n is a fixed number, the system parameters are computed offline as follows: $m = \text{OptimalM}()$, $s = \text{OptimalS}(m)$, $p = \text{OptimalP}(m, s)$, and T is set as the threshold value T^* before the last call to $\text{Potential}(m, s, p)$ is returned during the execution of $\text{OptimalM}()$. All four algorithms use bisection search (i.e., binary search), and therefore they are logarithmic in nature; see Appendix C. In our experiments, the average execution time for computing a set of optimal system parameters is 0.959 s on a DELL desktop with Pentium Dual-Core CPU of 3 GHz. If n is set based on historic data, we can precompute a table of system parameters with respect to different values of n, possibly in certain increments (e.g., 1000) to control the table size. In this way, if n changes significantly and thus the system parameters need to be updated, the new values can be fetched from the table.

V. Experiments

A. Experimental Setup

We evaluate the performance of ESC and compare it to existing work, including the Two-level Filtering Algorithm (TFA) [8], the Thresholded Bitmap Algorithm (TBA) [11], and the Compact Spread Estimator (CSE) [6]. TFA uses two filters to reduce both the number of sources to be monitored and the number of contacts to be stored. It is designed to satisfy the classification objective in (1). TBA is not designed for meeting the classification objective. It cannot ensure that false positive/negative ratios are bounded. CSE is designed to estimate the spreads of external sources in a very compact memory space. It can be used for spreader classification by reporting the sources whose estimated spreads exceed a certain threshold. However, the design of CSE makes it unsuitable for meeting the objective in (1).

Online Streaming Module (OSM) [5] is another related work. We do not implement OSM in this study because Yoon *et al.* show that, given the same amount of memory, CSE estimates spread values more accurately than OSM [6]. Moreover, the operations of OSM share certain similarity with Bloom filters. To store each contact, it performs three hash functions and makes three memory accesses. In comparison, ESC performs two hash functions and makes one memory access. Please refer to Section VII for more details on various spreader classification schemes.

The experiments use a real Internet traffic trace (inbound) captured by Cisco's Netflow at the main gateway of our campus for a week. For example, in one day of the week, the traffic trace records 10 702 677 distinct contacts, 4 007 256 distinct source IP addresses, and 56 167 distinct destination addresses. The average spread per source is 2.67, which means a source contacts 2.67 distinct destinations on average. Fig. 5 shows the number of sources with respect to the source spread in log scale. The number of sources decreases exponentially as the spread value increases from 1 to 500. After that, there is zero, one, or a few sources for each spread value.

We implement ESC, TFA, TBA, and CSE and execute them with the traffic trace as input. The application scenario is to monitor overall scanning activities by identifying most external sources that have suspiciously high spreads. Continuously gathering such information helps network administrators to learn the patterns of spread values and identify potential intruders that are frequently classified as heavy spreaders. In the experiments, the source of a contact is the IP address of the sender, and the destination is the IP address of the receiver. We have discussed in Section IV-C that there are two ways to define the measurement period. Accordingly, for our first set of experiments in Section V-B, we let each measurement period be a time interval in which 10 million contacts are received. For the rest of the experiments, the measurement period is one day. The experimental results are the average over the week-long data.

One performance metric used in comparison is the amount of memory that is required for a spreader classification scheme to meet a given classification objective. Remarkably, the number of bits required by ESC is far smaller than the number of distinct sources in the traffic trace. That is, ESC requires much less than 1 bit per source to perform spreader classification. Other performance metrics include the false positive ratio and the false negative ratio, which will be explained further shortly.

B. Comparison in Terms of Memory Requirement

The first set of experiments compares ESC and TFA for the amount of memory that they need in order to satisfy a given classification objective, which is specified by four parameters,

TABLE I
MEMORY REQUIREMENTS (IN MEGABYTES) OF ESC, TFA, AND ESC-1 (I.E., ESC WITH $p = 1$) WHEN $\alpha = 0.9$ AND $\beta = 0.1$

h	$l = 0.1h$			$l = 0.3h$			$l = 0.5h$			$l = 0.7h$		
	ESC	TFA	ESC-1	ESC	TFA	ESC-1	ESC	TFA	ESC-1	ESC	TFA	ESC-1
500	0.08	1.99	0.30	0.18	2.47	0.42	0.28	3.54	0.52	0.94	6.02	0.99
1000	0.07	1.00	0.25	0.09	1.18	0.30	0.14	1.76	0.40	0.42	3.05	0.82
2000	0.03	0.48	0.24	0.06	0.65	0.26	0.08	1.00	0.39	0.22	1.56	0.82
3000	0.03	0.40	0.23	0.03	0.50	0.26	0.05	0.62	0.39	0.15	1.01	0.82
4000	0.02	0.27	0.21	0.02	0.32	0.26	0.03	0.51	0.37	0.11	0.77	0.82
5000	0.01	0.23	0.20	0.02	0.29	0.25	0.02	0.40	0.37	0.10	0.60	0.82

TABLE II
MEMORY REQUIREMENTS (IN MEGABYTES) OF ESC, TFA, AND ESC-1 (I.E., ESC WITH $p = 1$) WHEN $\alpha = 0.95$ AND $\beta = 0.05$

h	$l = 0.1h$			$l = 0.3h$			$l = 0.5h$			$l = 0.7h$		
	ESC	TFA	ESC-1	ESC	TFA	ESC-1	ESC	TFA	ESC-1	ESC	TFA	ESC-1
500	0.12	2.38	0.35	0.19	3.15	0.43	0.43	4.44	0.62	1.45	8.01	1.55
1000	0.08	1.22	0.31	0.11	1.62	0.35	0.20	2.31	0.47	0.72	4.00	1.15
2000	0.03	0.66	0.25	0.07	0.82	0.30	0.11	1.15	0.43	0.35	2.05	1.16
3000	0.02	0.43	0.22	0.06	0.56	0.31	0.09	0.81	0.44	0.22	1.40	1.14
4000	0.02	0.33	0.22	0.04	0.41	0.30	0.06	0.60	0.44	0.20	1.01	1.15
5000	0.01	0.25	0.21	0.04	0.34	0.30	0.05	0.50	0.43	0.15	0.85	1.14

TABLE III
MEMORY REQUIREMENTS (IN MEGABYTES) OF ESC, TFA, AND ESC-1 (I.E., ESC WITH $p = 1$) WHEN $\alpha = 0.99$ AND $\beta = 0.01$

h	$l = 0.1h$			$l = 0.3h$			$l = 0.5h$			$l = 0.7h$		
	ESC	TFA	ESC-1	ESC	TFA	ESC-1	ESC	TFA	ESC-1	ESC	TFA	ESC-1
500	0.18	3.56	0.43	0.25	4.77	0.48	0.90	7.20	0.98	4.12	12.95	4.12
1000	0.09	1.88	0.32	0.14	2.34	0.36	0.46	3.46	0.61	1.50	6.45	3.00
2000	0.06	0.99	0.27	0.08	1.23	0.30	0.23	1.81	0.56	0.78	3.11	3.02
3000	0.04	0.63	0.23	0.06	0.82	0.31	0.15	1.22	0.59	0.51	2.10	3.02
4000	0.03	0.46	0.24	0.05	0.62	0.31	0.10	0.93	0.55	0.39	1.59	3.00
5000	0.03	0.40	0.22	0.05	0.52	0.31	0.09	0.72	0.56	0.32	1.31	3.00

α, β, h, and l. See Section II for the formal definition of the classification objective. We do not compare TBA and CSE here because they are not designed to meet this objective.

The memory required by ESC is determined based on the iterative algorithm in Section IV-C. The values of other parameters, s, T, and p, are also decided by the same algorithm. For example, when $\alpha = 0.9$, $\beta = 0.1$, $h = 5000$, $l = 0.7h$, $n^* = 10\text{M}$, the system parameters are $s = 40$, $T = 4250$, $p = 0.01$, and $m = 0.11$ MB. Using these parameters, we perform experiments on ESC with the traffic trace as input. The amount of memory required by TFA is determined experimentally based on the method in [8]. The parameters of TFA are chosen based on the original paper.

The memory requirements of ESC and TFA are presented in Tables I–III with respect to α, β, h, and l. For $\alpha = 0.9$ and $\beta = 0.1$, Table I shows that TFA requires 6–24 times of the memory that ESC requires, depending on the values of h and l (which the system administrator will select based on the organization's policy). For example, when $h = 500$ and $l = 0.5h$, ESC reduces the memory consumption by an order of magnitude when comparing with TFA.

Our experiments have also confirmed that the classification objective is indeed achieved by ESC. That is, the false positive ratio is always bounded by β, and the false negative ratio is bounded by $1 - \alpha$ for all experiments reported in Tables I–III.

To demonstrate the impact of probabilistic sampling, the table also includes the memory requirement of ESC when sampling is turned off (by setting $p = 1$). This version of ESC is denoted as ESC-1. Since p is set as a constant, the iterative algorithm in Section IV-C needs to be slightly modified: The procedure $\mathrm{OptimalP}(m, s)$ will always return 1, while other procedures remain the same. Table I shows that the memory saved by sampling is significant when h is large. For example, when $h = 5000$ and $l = 0.3h$, ESC with sampling uses less than 1/13 of the memory that is needed by ESC-1. However, when h becomes smaller or $\frac{l}{h}$ becomes larger, ESC has to choose a larger sampling probability in order to limit the error in spread estimation caused by sampling. Consequently, it has to store more contacts and thus require more memory. For instance, when $h = 500$ and $l = 0.5h$, ESC with sampling uses 55.6% of the memory that is needed by ESC-1.

Table II compares the memory requirements of ESC and TFA when $\alpha = 0.95$ and $\beta = 0.05$. Table III compares the memory requirements when $\alpha = 0.99$ and $\beta = 0.01$. They show similar results: 1) ESC uses significantly less memory than TFA; and 2) the probabilistic sampling method in ESC is critical for memory saving especially when h is large or $\frac{l}{h}$ is small. The tables also demonstrate that the memory requirement of either ESC or TFA increases when the classification objective becomes more stringent, i.e., α is set larger and β smaller.

TFA requires more memory because it stores the source and destination addresses of the contacts. In [28], the authors also indicate that Bloom filters [29], [30] can be used to reduce the memory consumption. However, the paper does not give detailed design or parameter settings. Therefore, we cannot implement the Bloom-filter version of TFA. The paper claims that the memory requirement will be reduced by a factor of 2.5 when Bloom filters are used. Even when this factor is taken into account in Tables I–III, memory saving by ESC will still be significant.

TABLE IV
FALSE NEGATIVE RATIO AND FALSE POSITIVE RATIO OF ESC, CSE, AND TBA WITH $m = 0.05$ MB

	FNR			FPR		
h	ESC	CSE	TBA	ESC	CSE	TBA
500	7.4e-2	0	2.6e-1	5.0e-2	1	9.0e-6
1000	1.0e-2	0	2.6e-1	5.5e-3	1	9.0e-6
2000	4.2e-3	0	2.5e-1	2.0e-3	1	1.1e-5
3000	5.5e-3	0	2.5e-1	2.0e-3	1	1.0e-5
4000	0	0	2.4e-1	2.0e-3	1	7.0e-6
5000	0	0	2.4e-1	2.0e-3	1	7.0e-6

TABLE V
FALSE NEGATIVE RATIO AND FALSE POSITIVE RATIO OF ESC, CSE, AND TBA WITH $m = 0.2$ MB

	FNR			FPR		
h	ESC	CSE	TBA	ESC	CSE	TBA
500	1.2e-2	3.3e-2	3.7e-3	1.5e-3	1.2e-1	1.8e-4
1000	8.8e-4	0	3.7e-3	7.5e-4	5.5e-2	1.9e-4
2000	0	0	9.3e-3	7.5e-4	5.5e-2	2.0e-4
3000	0	0	7.4e-3	7.5e-4	5.5e-2	1.8e-4
4000	0	0	1.9e-3	7.5e-4	5.5e-2	1.9e-4
5000	0	0	3.7e-3	7.5e-4	5.5e-2	1.8e-4

C. *Comparison in Terms of False Positive Ratio and False Negative Ratio*

The *false positive ratio* (FPR) is defined as the fraction of all non-heavy spreaders (whose spreads are no more than l) that are mistakenly reported as heavy spreaders. The *false negative ratio* (FNR) is the fraction of all heavy spreaders (whose spreads are no less than h) that are not reported by the system. In Section V-B, we have shown that, given the bounds of FPR and FNR, it takes ESC much less memory to achieve the bounds than TFA. Since CSE and TBA are not designed for meeting a given set of bounds, we compare our ESC to them by a different set of experiments that measure and compare the FPR and FNR values under a fixed amount of SRAM.

Given a fixed memory size m, we use $\mathrm{OptimalS}(m, s)$ in Section IV-C to determine the value of s in ESC, use $\mathrm{OptimalP}(m, s)$ to determine the value of p, and then set the threshold T as $\frac{h+l}{2}$. We perform experiments using the week-long traffic trace. We average the daily FPR and FNR values over the week. For $m = 0.05$ MB, and 0.2 MB, the results are presented in Tables IV and V, respectively. In both tables, $l = 0.5h$. We also perform the same experiments for CSE and TBA, and the results are presented in the tables as well. The optimal parameters are chosen for each scheme based on the original papers.

When the available memory is very small, such as 0.05 MB in Table IV, CSE has zero FNR, but its FPR is 1.0, which means it reports all non-heavy spreaders. The reason is that, without probabilistic sampling, CSE stores information of too many contacts such that its data structure is fully saturated. In this case, the spread estimation method of CSE breaks down. TBA has a small FPR but its FNR is large. For example, when $h = 500$, its FNR is 26%. Only ESC achieves small values for both FNR and FPR. For example, when $h = 500$, its FNR is 7.4% and its FPR is 5.0%. These values decrease quickly as h increases. When $h = 1000$, they are 1.0% and 0.55%, respectively, while the FNR of TBA remains to be 26%. When the available memory increases in Table V, the performance of all three schemes improves. Still, ESC performs better in most cases.

TABLE VI
FALSE NEGATIVES RATIO AND FALSE POSITIVES RATIO WITH $\alpha = 0.95$ AND $\beta = 0.05$

	$l = 0.3h$		$l = 0.5h$		$l = 0.7h$	
h	FNR	FPR	FNR	FPR	FNR	FPR
500	4.4e-3	3.6e-6	2.0e-3	1.0e-6	3.2e-4	0
1000	3.7e-3	3.0e-6	1.9e-3	1.0e-6	5.6e-4	0
2000	4.2e-3	1.0e-6	2.1e-3	1.0e-6	1.1e-3	1e-6
3000	1.8e-2	0	0	0	1.6e-3	0
4000	1.4e-2	1.0e-6	0	0	7.1e-3	0
5000	3.1e-3	0	6.4e-3	1.0e-6	0	1e-6

D. *Performance of ESC*

The iterative algorithm in Section IV-C gives the memory requirement for meeting a certain classification objective in the worst case. In reality, the worst-case scenario rarely happens. Hence, we expect the observed FPR to be much smaller than β and the observed FNR to be much smaller than $(1 - \alpha)$. This is indeed what we see in our experiments.

We run ESC on the week-long traffic trace under the following parameter settings: $\alpha = 0.95$, $\beta = 0.05$, h is varied from 500 to 5000, and l is varied from $0.3h$, $0.5h$ to $0.7h$. We use the iterative algorithm to determine the values of m and other system parameters. (Unlike Section V-C, m is not a given value.) We then collect the daily values of FPR and FNR, and the average results are shown in Table VI. It shows that the real FPR/FNR are much smaller than the 5% objective (i.e., $1 - \alpha$ or β).

The reason is that the amount of memory m that ESC uses is determined based on the worst-case scenario, where the spreads of all non-heavy spreaders are l and the spreads of all heavy spreaders are h. Recall that in Section IV-B, we choose $k = l$ in (10) and $k = h$ in (9). However, in reality, not all non-heavy spreaders make the same number l of distinct contacts; many sources make very small numbers of contacts, as shown in Fig. 5. For a non-heavy spreader whose spread is much smaller than l, the probability for its estimated spread to exceed the threshold T (thus resulting in false positive) is certainly smaller than that of a non-heavy spreader whose spread is l. Similarly, not all heavy spreaders make the same number h of distinct contacts. For a heavy spreader whose spread is much larger than h, the probability for its estimated spread to fall below the threshold (which results in false negative) is certainly smaller than that of a heavy spreader whose spread is h.

The above observation is also confirmed by experiment. We set $\alpha = 95\%$, $\beta = 5\%$, $h = 1000$, and $l = 0.5h$. After executing ESC on the traffic trace, we count the average daily number of false positives for sources whose spreads are in the range of $(0, l/5], (l/5, 2l/5], \ldots,$ or $(4l/5, l]$, and the average daily number of false negatives for sources whose spreads are in the range of $[1000, 1200), [1200, 1400) \ldots$ The resulting FPR/FNR values in those ranges are presented in Fig. 6, where each point represents FPR (or FNR) for in a certain range of spread values (on the horizontal axis). The figure shows that FPR decreases quickly to zero for sources whose spreads are very small and FNR decreases quickly to zero for sources whose spreads are very large.

VI. MULTI-OBJECTIVE SPREADER CLASSIFICATION

So far, we have studied spreader classification with a single objective: for example, reporting sources whose spreads are 100

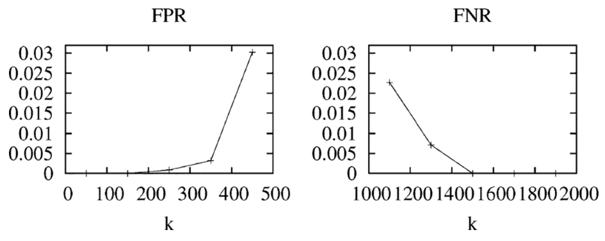


Fig. 6. False positive ratio and false negative ratio with $h = 1000$ and $l = 0.5h$.

or more with at least 95% probability, while reporting sources whose spreads are 75 or less with at most 5% probability. In practice, one may want to configure a router with more than one objective. For instance, in addition to the above objective of reporting *modestly heavy spreaders*, we may want to add another objective to identify more aggressive ones: reporting sources whose spreads are 1000 or more as *heavy spreaders* with at least 99% probability, and reporting sources whose spreads are 500 or less with at most 1% probability. How to efficiently perform spreader classification with multiple objectives is the subject of this section.

A. Problem Definition

Suppose there are t objectives. The jth objective, for any $j \in [1,t]$ is defined by four parameters, h_j, l_j, α_j, and β_j. Among them, h_j and l_j are positive integers with $h_j > l_j$, while α_j and β_j are probability values, $0 < \alpha_j < 1$ and $0 < \beta_j < 1$. The jth objective is to report any source whose spread is h_j or larger as a type-j heavy spreader with a probability no less than α_j, and report any source whose spread is l_j or smaller with a probability no more than β_j. Similar to (1), the conditional probabilities that express the whole set of objectives are

$$\begin{gathered}
\text{Prob}\{\text{report } src \text{ as a type-1 heavy spreader} \,|\, k \geq h_1\} \geq \alpha_1 \\
\text{Prob}\{\text{report } src \text{ as a type-1 heavy spreader} \,|\, k \leq l_1\} \leq \beta_1 \\
\cdots \\
\text{Prob}\{\text{report } src \text{ as a type-}j \text{ heavy spreader} \,|\, k \geq h_j\} \geq \alpha_j \\
\text{Prob}\{\text{report } src \text{ as a type-}j \text{ heavy spreader} \,|\, k \leq l_j\} \leq \beta_j \\
\cdots \\
\text{Prob}\{\text{report } src \text{ as a type-}t \text{ heavy spreader} \,|\, k \geq h_t\} \geq \alpha_t \\
\text{Prob}\{\text{report } src \text{ as a type-}t \text{ heavy spreader} \,|\, k \leq l_t\} \leq \beta_t
\end{gathered} \tag{15}$$

where src is an arbitrary source and its spread is k. Each objective is expressed by a pair of conditional probabilities. Our goal is to minimize the amount of SRAM that is needed for achieving the above objectives.

B. Multi-Objective Spreader Classification Scheme and Optimal System Parameters

The spreader classification scheme in Section III can be extended to detect heavy spreaders under multiple objectives. The probabilistic sampling, dynamic bit sharing, and maximum likelihood-based spread estimation stay the same. The difference is how to determine the optimal system parameters. For spreader classification with one objective, we determine a threshold value T in Section IV and report all sources whose estimated spreads exceed the threshold. The optimal values of T, s, p, and m are computed based on (11). For spreader classification with multiple objectives, we need to determine one threshold for each objective. If the estimated spread of a source exceeds the threshold T_j for the jth objective, the source is reported as a type-j heavy spreader, where $1 \leq j \leq t$.

Following a mathematical process similar to Section IV, we can derive the set of constraints that the system parameters must satisfy in order to meet all objectives. We want to minimize the amount of memory that is needed to satisfy the constraints. (We omit the derivation process to avoid excessive repetition of content similar to Section IV.)

$$\text{Minimize} \quad m$$
$$\text{Subject to} \quad \forall 1 \leq j \leq t \tag{16}$$
$$\sum_{i=0}^{\lfloor C_j \rfloor} \binom{s}{i} \cdot q(h_j)^i \cdot (1 - q(h_j))^{s-i} \geq \alpha_j \tag{17}$$
$$\sum_{i=0}^{\lfloor C_j \rfloor} \binom{s}{i} \cdot q(l_j)^i \cdot (1 - q(l_j))^{s-i} \leq \beta_j \tag{18}$$

where

$$C_j = s \cdot \left(1 - \frac{p}{m}\right)^n \cdot \left(\frac{1 - \frac{p}{s}}{1 - \frac{p}{m}}\right)^{T_j}.$$

The conditional probabilities for the jth objective are transformed into two equivalent constraints: $\text{Prob}\{\text{report } src \,|\, k \geq h_j\} \geq \alpha_j$ is transformed to (17) and $\text{Prob}\{\text{report } src \,|\, k \leq l_j\} \leq \beta_j$ to (18). Our goal is to determine the optimal values for T_j $(1 \leq j \leq t)$, s and p, such that they together minimize m. Note that there exists one threshold for each objective. In total, there are t thresholds. However, s, p, and m are common parameters shared for all objectives.

The algorithm in Section IV-C can be extended to solve the above constrained optimization problem. We briefly describe the solution below. Consider the left side of (17) as a function $F_{h_j}(m,s,p,T_j)$ and the left side of (18) as a function $F_{l_j}(m,s,p,T_j)$. Namely, $\forall 1 \leq j \leq t$

$$F_{h_j}(m,s,p,T_j) = \sum_{i=0}^{\lfloor C_j \rfloor} \binom{s}{i} \cdot q(h_j)^i \cdot (1 - q(h_j))^{s-i}$$
$$F_{l_j}(m,s,p,T_j) = \sum_{i=0}^{\lfloor C_j \rfloor} \binom{s}{i} \cdot q(l_j)^i \cdot (1 - q(l_j))^{s-i}.$$

We modify the iterative numerical algorithm in Section IV-C to determine the optimal system parameters. The revised algorithm consists of five procedures, which are described as follows.

- First, we overload the procedure $\text{Potential}(m,s,p)$ in Section IV-C and add one input parameter, j, indicating which functions the procedure is applied to. More specifically, $\forall j \in [1,t]$, the new procedure $\text{Potential}(m,s,p,j)$ is applied to functions $F_{h_j}(m,s,p,T_j)$ and $F_{l_j}(m,s,p,T_j)$. It uses the same binary search method as in Algorithm 1 to find the optimal value of T_j that maximizes the value of $F_{h_j}(m,s,p,T_j)$ under the condition that $F_{l_j}(m,s,p,T_j) \leq \beta$. The procedure returns the maximum value of $F_{h_j}(m,s,p,T_j)$, and as a byproduct, determines the optimal value of T_j. The pseudocode is the same as Algorithm 1 in Appendix C except that $F_h(m,s,p,T)$ is replaced by $F_{h_j}(m,s,p,T_j)$ and $F_l(m,s,p,T)$ is replaced by $F_{l_j}(m,s,p,T_j)$.
- Second, we construct a new procedure called $\text{PotentialAll}(m,s,p)$, which takes a value of m,

a value of s and a value of p as input and returns the minimum value of $\{\text{Potential}(m,s,p,1) - \alpha_1, \text{Potential}(m,s,p,2) - \alpha_2, \ldots, \text{Potential}(m,s,p,t) - \alpha_t\}$. Clearly, if $\text{PotentialAll}(m,s,p) \geq 0$, all objectives can be satisfied.

Essentially, the value of $\text{PotentialAll}(m,s,p)$ quantitatively indicates how conservative or aggressive we have chosen the value of m. If $\text{PotentialAll}(m,s,p)$ is positive, it means that the performance achieved by current memory size is more than required. We shall reduce m. On the other hand, if $\text{PotentialAll}(m,s,p)$ is negative, we shall increase m.

- Third, given a value of m and a value of s, we construct a procedure $\text{OptimalP}'(m,s)$ that determines the optimal value p^* such that $\text{PotentialAll}(m,s,p^*)$ is maximized. This procedure is similar to its counterpart in Section IV-C. The pseudocode is the same as Algorithm 2 in Appendix C except that $\text{Potential}(m,s,p)$ is replaced with $\text{PotentialAll}(m,s,p)$.
- Fourth, given a value of m, we construct a procedure $\text{OptimalS}'(m)$ that determines the optimal value s^* such that $\text{PotentialAll}(m,s^*,\text{OptimalP}'(m,s^*))$ is maximized. This procedure is the same as its counterpart in Section IV-C (Algorithm 3 in Appendix C), except that $\text{Potential}(\ldots)$ is replaced with $\text{PotentialAll}(\ldots)$ and $\text{OptimalP}(m,s)$ is replaced with $\text{OptimalP}'(m,s)$.
- Fifth, we construct a procedure $\text{OptimalM}'()$ that determines the minimum memory requirement m^* such that $\text{PotentialAll}(m, \text{OptimalS}'(m), \text{OptimalP}'(m, \text{OptimalS}'(m))) \geq 0$ is satisfied. Again, this procedure is similar as its counterpart in Section IV-C (Algorithm 4 in Appendix C). We skip the detailed description, which is virtually identical to the description in Section IV-C.

In practice, given the objectives that are specified by $t, \alpha_j, \beta_j, h_j$ and l_j, $1 \leq j \leq t$, a network administrator sets $m = \text{OptimalM}'(), s = \text{OptimalS}'(m), p = \text{OptimalP}'(m,s)$, and $T_j (1 \leq j \leq t)$ as the threshold value before the last call of $\text{Potential}(m,s,p,j)$ is returned during the execution of $\text{OptimalM}'()$.

C. Additional Experimental Results

We perform additional experiments to evaluate our multi-objective spreader classification scheme. We use the same traffic trace as described in Section V. We do not implement TFA, TBA, or CSE because none of them can be applied for multi-objective spreader classification. In the new experiments, we let $t = 2$, i.e., there are two objectives, which are specified as follows: $h_1 = 500$, $\alpha_1 = 0.9$, $\beta_1 = 0.1$, $h_2 = 5000$, $\alpha_2 = 0.99$, and $\beta = 0.01$.

In the first set of experiments, we let $l_2 = 0.5h_2$ and vary l_1 from $0.1h_1$, $0.3h_1$, $0.5h_1$, to $0.7h_1$. We use the iterative algorithm in Section VI-B to compute the minimum amount of memory needed, as well as the optimal values for other system parameters. After that, we perform experiments based on these system parameters to report the heavy spreaders in the traffic trace. We measure the FPR and FNR values that are observed in the experiments. The results are presented in Table VII. The FNR values for type-1 heavy spreaders are shown in the column labeled with FNR1; they are indeed smaller than $1 - \alpha_1$. The FPR values for type-1 heavy spreaders are shown in the column labeled with FPR1; they are smaller than β_1 as required. Similarly, the data in columns FNR2 and FPR2 show that the second objective (specified by α_2 and β_2) is also met.

TABLE VII
MEMORY REQUIREMENT, FALSE NEGATIVE RATIO, AND FALSE POSITIVE RATIO WITH $h_1 = 500$, $h_2 = 5000$, AND $l_2 = 0.5h_2$

		$\alpha_1 = 0.9, \beta_1 = 0.1$		$\alpha_2 = 0.99, \beta_2 = 0.01$	
l_1	m(MB)	FNR1	FPR1	FNR2	FPR2
$0.1h_1$	0.2	5.0e-3	1.7e-2	0	0
$0.3h_1$	0.3	8.0e-3	4.0e-4	0	0
$0.5h_1$	0.4	4.0e-3	1.0e-6	0	0
$0.7h_1$	1.1	2.0e-3	1.0e-6	0	0

TABLE VIII
MEMORY REQUIREMENT, FALSE NEGATIVE RATIO, AND FALSE POSITIVE RATIO WITH $h_1 = 500$, $h_2 = 5000$, AND $l_1 = 0.5h_1$

		$\alpha_1 = 0.9, \beta_1 = 0.1$		$\alpha_2 = 0.99, \beta_2 = 0.01$	
l_2	m(MB)	FNR1	FPR1	FNR2	FPR2
$0.1h_2$	0.3	1.5e-2	8.7e-5	0	0
$0.3h_2$	0.4	4.0e-3	2.0e-6	0	0
$0.5h_2$	0.4	4.0e-3	3.0e-6	0	3.0e-6
$0.7h_2$	0.6	5.0e-3	5.0e-6	0	1.1e-5

In the second set of experiments, we let $l_1 = 0.5h_1$ and vary l_2 from $0.1h_2$, $0.3h_2$, $0.5h_2$, to $0.7h_2$. Again we use the iterative algorithm to determine the system parameters and run experiments to measure the FPR and FNR values. The results are presented in Table VIII. The data are interpreted in a similar way as we do for Table VII. Clearly, both objectives are met.

VII. Related Work

Venkataraman *et al.* [8] use hash tables to store the addresses of the sampled contacts. Their main contribution is to derive the optimal sampling probability that achieves a classification objective with prespecified upper bounds on false positive ratio and false negative ratio. However, because their algorithms store the contact addresses, it leaves great room for improvement. Even if Bloom filters are used, the room for improvement is still significant, as we have argued in Section V.

Estan *et al.* [10] propose a variety of bitmap algorithms to store the contacts (or active flows in their context). It saves space because each destination address is stored as a bit. However, assigning one bitmap to each source is not cheap if the average number of contacts per source is small. In addition, 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. Cao *et al.* [11] develop the thresholded bitmap algorithm based on the virtual bitmap algorithm presented in [10] for spread estimation. They use probabilistic sampling to reduce the information to be stored.

Zhao *et al.* [5] share a set of bitmaps among all sources. The scheme assigns three pseudorandomly selected bitmaps to each source. When the source contacts a destination, the destination is stored by setting one bit in each of the three bitmaps. Because the bitmaps are shared by others, the information stored for one source becomes noise for others. Yoon *et al.* [6] observe that the noise introduced by sharing bitmaps cannot be appropriately removed if the number of bitmaps is not sufficiently large. By sharing bits instead of bitmaps, CSE considerably reduces the memory consumption.

Also related is the work by Bandi *et al.* [27] on the heavy distinct hitter problem, which is essentially the same as spreader classification. Their algorithm exploits TCAM, a special kind of memory found in network processing units (NPUs). The emphasis of their work is on the processing time.

A related branch of research is the detection of heavy-hitters [19]–[26]. A heavy-hitter is a source that sends a lot of packets during a measurement period no matter whether the packets are sent to a few or many distinct destinations.

VIII. Conclusion

Spreader classification is one of the most important functions in intrusion detection systems. The recent research trend is to implement such a function in the tight SRAM space to catch up with the rapid advance in network speed. This paper proposes an efficient scheme based on a new method for spreader classification, which optimally combines probabilistic sampling, bit-sharing storage, and maximum likelihood estimation. We demonstrate theoretically and experimentally that the new scheme is able to achieve a classification objective with arbitrarily set bounds on worst-case false positive/negative ratios. It does so in a very tight memory space where the number of bits available is much smaller than the number of external sources to be monitored. In addition, we extend our scheme to solve the multi-objective spreader classification problem.

Appendix A
Compute $\hat{k}$

In the standard process of maximum likelihood estimation, the unknown value k is technically treated as a variable in (4). We want to find an estimate $\hat{k}$ that maximizes the likelihood function. Namely

$$\hat{k} = \underset{k}{\arg\max}\{L\}. \tag{19}$$

Since the maxima is not affected by monotone transformations, we use logarithm to turn the right side of (4) from product to summation

$$\ln(L) = U_s \cdot \ln(q(k)) + (s - U_s) \cdot \ln(1 - q(k)).$$

From (3), the above equation can be written as

$$\begin{aligned}\ln(L) = U_s &\left((n-k)\ln\left(1-\frac{p}{m}\right) + k\ln\left(1-\frac{p}{s}\right)\right)\\ &+(s-U_s)\cdot\ln\left(1-\left(1-\frac{p}{m}\right)^{n-k}\left(1-\frac{p}{s}\right)^{k}\right).\end{aligned}$$

To find the maxima, we differentiate both sides

$$\frac{\partial \ln(L)}{\partial k} = \ln\left(\frac{1-\frac{p}{s}}{1-\frac{p}{m}}\right) \cdot \frac{U_s - s\left(1-\frac{p}{m}\right)^{n-k}\left(1-\frac{p}{s}\right)^{k}}{1-\left(1-\frac{p}{m}\right)^{n-k}\left(1-\frac{p}{s}\right)^{k}}. \tag{20}$$

We then let the right side be zero. That is

$$U_s = s\left(1-\frac{p}{m}\right)^{n-k}\left(1-\frac{p}{s}\right)^{k}. \tag{21}$$

Taking logarithm on both sides, we have

$$\begin{aligned}\ln\frac{U_s}{s} &= n\ln\left(1-\frac{p}{m}\right) + k\left(\ln\left(1-\frac{p}{s}\right) - \ln\left(1-\frac{p}{m}\right)\right)\\ k &= \frac{\ln V_s - n\ln\left(1-\frac{p}{m}\right)}{\ln\left(1-\frac{p}{s}\right)-\ln\left(1-\frac{p}{m}\right)}.\end{aligned} \tag{22}$$

Suppose the number of sources (which equals to the number of logical bitmaps) is sufficiently large. Because every bit in every logical bitmap is randomly selected from B, in this sense, each of the n contacts has about the same probability $\frac{p}{m}$ of setting any bit in B. Hence, we have

$$E(V_m) = \left(1-\frac{p}{m}\right)^n. \tag{23}$$

Applying (23) to (22), we have

$$k = \frac{\ln V_s - \ln E(V_m)}{\ln\left(1-\frac{p}{s}\right)-\ln\left(1-\frac{p}{m}\right)}. \tag{24}$$

Replacing $E(V_m)$ by the instance value V_m, we have the estimation for k in (2). V_s can be measured by counting the number of zeros in $LB(src)$, V_m can be measured by counting the number of zeros in B, and s, p, and m are preset parameters of ESC.

Appendix B
Variance of V_m

Let A_i be the event that the ith bit in B remains "0" at the end of the measurement period and 1_{A_i} be the corresponding indicator random variable. Let U_m be the random variable for the number of "0" bits in B. We first derive the probability for A_i to occur and the expected value of U_m. For an arbitrary bit in B, each distinct contact has a probability of $\frac{p}{m}$ to set the bit to one. All contacts are independent of each other when setting bits in B. Hence

$$\text{Prob}\{A_i\} = \left(1-\frac{p}{m}\right)^n \qquad \forall i \in [0, s).$$

The probability for A_i and A_j, $\forall i,j \in [0,m), i \neq j$, to happen simultaneously is

$$\text{Prob}\{A_i \cap A_j\} = \left(1-\frac{2p}{m}\right)^n.$$

Since $V_m = \frac{U_m}{m}$ and $U_m = \sum_{i=1}^m 1_{A_i}$, we have

$$\begin{aligned}E\left(V_m^2\right) &= \frac{1}{m^2}E\left(\left(\sum_{i=1}^m 1_{A_i}\right)^2\right)\\ &= \frac{1}{m^2}E\left(\sum_{i=1}^m 1_{A_i}^2\right) + \frac{2}{m^2}E\left(\sum_{1\le i<j\le m} 1_{A_i}1_{A_j}\right)\\ &= \frac{1}{m}\left(1-\frac{p}{m}\right)^n + \frac{m-1}{m}\left(1-\frac{2p}{m}\right)^n.\end{aligned}$$

Based on (23) and the equation above, we have

$$\begin{aligned} Var(V_m) &= E\left(V_m^2\right) - E(V_m)^2 \\ &= \frac{1}{m}\left(1-\frac{p}{m}\right)^n + \frac{m-1}{m}\left(1-\frac{2p}{m}\right)^n \\ &\quad - \left(1-\frac{p}{m}\right)^{2n} \\ &\simeq \frac{e^{-\frac{np}{m}}\left(1-\left(1+\frac{np^2}{m}\right)e^{-\frac{np}{m}}\right)}{m}. \end{aligned} \tag{25}$$

APPENDIX C
ALGORITHMS FOR OPTIMAL SYSTEM PARAMETERS

Algorithm 1: $\text{Potential}(m, s, p)$

```
INPUT: $m, s, p$, and $\beta$
OUTPUT: The maximum value of $F_h(m, s, p, T)$ under the condition that $F_l(m, s, p, T) \leq \beta$.

  Pick a small integer $T_1$ such that $F_l(m, s, p, T_1) > \beta$ and
  a large integer $T_2$ such that $F_l(m, s, p, T_2) \leq \beta$
  while $T_2 - T_1 > 1$ do
    $\bar{T} = \lfloor (T_1 + T_2)/2 \rfloor$
    if $F_l(m, s, p, \bar{T}) \leq \beta$ then
      $T_1 = \bar{T}$
    else
      $T_2 = \bar{T}$
    end if
  end while
  $T^* = \bar{T}$
  return $F_h(m, s, p, T^*)$
```

Algorithm 2: $\text{OptimalP}(m, s)$

```
INPUT: $m, s$, and $\delta$
OUTPUT: The optimal value of $p^*$ such that $\text{Potential}(m, s, p^*)$ is maximized

  $p_1 = 0, p_2 = 1$
  while $p_2 - p_1 > \delta$ do
    $\bar{p} = (p_1 + p_2)/2$
    if $\text{Potential}(m, s, \bar{p}) < \text{Potential}(m, s, \bar{p} + \delta)$ then
      $p_1 = \bar{p}$
    else
      $p_2 = \bar{p}$
    end if
  end while
  $p^* = (p_1 + p_2)/2$
  return $p^*$
```

Algorithm 3: $\text{OptimalS}(m)$

```
INPUT: $m$
OUTPUT: The optimal value of $s^*$ such that $\text{Potential}(m, s^*, \text{OptimalP}(m, s^*))$ is maximized

  $s_1 = 1, s_2 = m$
  while $s_2 - s_1 > 1$ do
    $\bar{s} = \lfloor (s_1 + s_2)/2 \rfloor$
    if $\text{Potential}(m, \bar{s}, \text{OptimalP}(m, \bar{s})) <$
    $\text{Potential}(m, \bar{s} + 1, \text{OptimalP}(m, \bar{s} + 1)$ then
      $s_1 = \bar{s}$
    else
      $s_2 = \bar{s}$
    end if
  end while
  $s^* = \bar{s}$
  return $s^*$
```

Algorithm 4: $\text{OptimalM}()$

```
OUTPUT: The smallest value $m^*$ that satisfies $\text{Potential}(m^*, \ldots) \geq \alpha$

  Pick a small value $m_1$ such that $\text{Potential}(m_1, \ldots) \leq \alpha$
  and a large value $m_2$ such that $\text{Potential}(m_2, \ldots) \geq \alpha$.
  while $m_2 - m_1 > 0$ do
    $\bar{m} = \lfloor (m_1 + m_2)/2 \rfloor$
    if $\text{Potential}(\bar{m}, \ldots) \leq \alpha$ then
      $m_1 = \bar{m}$
    else
      $m_2 = \bar{m}$
    end if
  end while
  $m^* = \bar{m}$
  return $m^*$
```

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their constructive comments.

REFERENCES

[1] W. D. Gardner, "Researchers transmit optical data at 16.4 Tbps," *Inf. Week*, Feb. 2008 [Online]. Available: http://www.informationweek.com/researchers-transmit-optical-data-at-164/206900603

[2] H. Song, F. Hao, M. Kodialam, and T. Lakshman, "IPv6 lookups using distributed and load balanced bloom filters for 100 Gbps core router line cards," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 2518–2526.

[3] C. Hermsmeyer, H. Song, R. Schlenk, R. Gemelli, and S. Bunse, "Towards 100G packet processing: Challenges and technologies," *Bell Labs Tech. J.*, vol. 14, no. 2, pp. 57–80, 2009.

[4] Y. Lu, A. Montanari, B. Prabhakar, S. Dharmapurikar, and A. Kabbani, "Counter braids: A novel counter architecture for per-flow measurement," in *Proc. ACM SIGMETRICS*, Jun. 2008, pp. 121–132.

[5] Q. Zhao, J. Xu, and A. Kumar, "Detection of super sources and destinations in high-speed networks: Algorithms, analysis and evaluation," *IEEE J. Sel. Areas Commun.*, vol. 24, no. 10, pp. 1840–1852, Oct. 2006.

[6] M. Yoon, T. Li, S. Chen, and J. Peir, "Fit a spread estimator in small memory," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 504–512.

[7] Q. Zhao, J. Xu, and Z. Liu, "Design of a novel statistics counter architecture with optimal space and time efficiency," in *Proc. ACM SIGMETRICS/Perform.*, 2006, pp. 323–334.

[8] S. Venkatataman, D. Song, P. Gibbons, and A. Blum, "New streaming algorithms for fast detection of superspreaders," in *Proc. NDSS*, Feb. 2005, p. 6.

[9] K. Hwang, B. Vander-Zanden, and H. Taylor, "A linear-time probabilistic counting algorithm for database applications," *Trans. Database Syst.*, vol. 15, no. 2, pp. 208–229, Jun. 1990.

[10] C. Estan, G. Varghese, and M. Fish, "Bitmap algorithms for counting active flows on high-speed links," *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, pp. 925–937, Oct. 2006.

[11] J. Cao, Y. Jin, A. Chen, T. Bu, and Z. Zhang, "Identifying high cardinality internet hosts," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 810–818.

[12] S. Staniford, J. Hoagland, and J. McAlerney, "Practical automated detection of stealthy portscans," *J. Comput. Secur.*, vol. 10, pp. 105–136, 2002.

[13] D. Plonka, "FlowScan: A network traffic flow reporting and visualization tool," in *Proc. USENIX LISA*, 2000, pp. 305–318.

[14] T. Li, S. Chen, and Y. Ling, "Fast and compact per-flow traffic measurement through randomized counter sharing," in *Proc. IEEE INFOCOM*, 2011, pp. 1799–1807.

[15] D. Shah, S. Iyer, B. Prabhakar, and N. McKeown, "Maintaining statistics counters in router line cards," *IEEE Micro*, vol. 22, no. 1, pp. 76–81, Jan.–Feb. 2002.

[16] S. Ramabhadran and G. Varghese, "Efficient implementation of a statistics counter architecture," in *Proc. ACM SIGMETRICS*, Jun. 2003, pp. 261–271.

[17] Y. Lu and B. Prabhakar, "Robust counting via counter braids: An error-resilient network measurement architecture," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 522–530.

[18] T. Li, S. Chen, and Y. Qiao, "Origin-destination flow measurement in high-speed networks," in *Proc. IEEE INFOCOM, Mini-Conf.*, 2012, pp. 2526–2530.

[19] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Proc. ICALP*, Jul. 2002, p. 784.

[20] G. Cormode and S. Muthukrishnan, "Space efficient mining of multigraph streams," in *Proc. ACM PODS*, Jun. 2005, pp. 271–282.

[21] E. Demaine, A. Lopez-Ortiz, and J. Ian-Munro, "Frequency estimation of internet packet streams with limited space," in *Proc. ESA*, Sep. 2002, pp. 348–360.

[22] X. Dimitropoulos, P. Hurley, and A. Kind, "Probabilistic lossy counting: An efficient algorithm for finding heavy hitters," *Comput. Commun. Rev.*, vol. 38, no. 1, pp. 7–16, 2008.

[23] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proc. ACM SIGCOMM*, Aug. 2002, pp. 323–336.

[24] P. Gibbons and Y. Matias, "New sampling-based summary statistics for improving approximate query answers," in *Proc. ACM SIGMOD*, Jun. 1998, pp. 331–342.

[25] G. Manku and R. Motwani, "Approximate frequency counts over data streams," in *Proc. VLDB*, Aug. 2002, pp. 346–357.

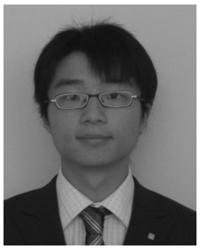
[26] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund, "Online identification of hierarchical heavy hitters: Algorithms, evaluation, and application," in *Proc. ACM SIGCOMM IMC*, Oct. 2004, pp. 101–114.

[27] N. Bandi, D. Agrawal, and A. Abbadi, "Fast algorithms for heavy distinct hitters using associative memories," in *Proc. IEEE ICDCS*, Jun. 2007, p. 6.

[28] Q. Zhao, A. Kumar, and J. Xu, "Joint data streaming and sampling techniques for detection of super sources and destinations," in *Proc. USENIX/ACM Internet Meas. Conf.*, Oct. 2005, p. 7.

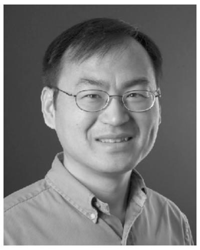
[29] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[30] A. Broder and M. Mitzenmacher, "Network applications of Bloom filters: A survey," *Internet Math.*, vol. 1, no. 4, pp. 485–509, 2002.



Tao Li received the B.S. degree in computer science from the University of Science and Technology of China, Hefei, China, in 2007, and is currently pursuing the Ph.D. degree in computer and information science and engineering at the University of Florida, Gainesville.

His advisor is Dr. Shigang Chen, and his research interests include network security and sensor networks.



Shigang Chen (M'04–SM'12) received the B.S. degree from the University of Science and Technology of China, Hefei, China, in 1993, and the M.S. and Ph.D. degrees from the University of Illinois at Urbana–Champaign in 1996 and 1999, respectively, all in computer science.

He is an Associate Professor with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville. After graduation, he worked with Cisco Systems, San Jose, CA, for three years before joining the University of Florida in 2002. He served on the technical advisory board for Protego Networks from 2002 to 2003. He published more than 100 peer-reviewed papers, with about 5000 citations. He holds 11 US patents. His research interests include computer networks, Internet security, wireless communications, and distributed computing.

Dr. Chen is an Associate Editor of the IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, and *Computer Networks*. He also served as a Guest Editors for *Advances in Multimedia* in 2007, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY in 2005, and for *Wireless Networks* in 2005. He has been serving on the Steering Committee of IEEE IWQoS since 2010. He received the IEEE Communications Society Best Tutorial Paper Award in 1999 and the NSF CAREER Award in 2007.

Wen Luo received the B.S. degree in computer science and technology from the University of Science and Technology of China, Hefei, China, in 2008, and is currently pursuing the Ph.D. degree in computer and information science and engineering at the University of Florida, Gainesville.

His advisor is Dr. Shigang Chen, and his research interests include RFID technologies and Internet traffic measurement.

Ming Zhang (S'10) received the B.S. degree in computer science from Sun Yat-sen University, Guangzhou, China, in 2000, the M.S. degree in computer science from Peking University, Beijing, China, in 2004, and the Ph.D. degree in computer engineering from the University of Florida, Gainesville, in 2010.

His research interests include real-time communication and security in wireless sensor networks, fairness in wireless LANs, and throughput in large RFID systems.

Yan Qiao received the B.S. degree in computer science and technology from Shanghai Jiao Tong University, Shanghai, China, in 2009, and is currently pursuing the Ph.D. degree in computer and information science and engineering at the University of Florida, Gainesville.

Her advisor is Dr. Shigang Chen. Her research interests include network measurement, algorithms, and RFID protocols.