This paper was presented as part of the main technical program at IEEE INFOCOM 2011

Scan Detection in High-Speed Networks Based on Optimal Dynamic Bit Sharing

Tao Li Shigang Chen Wen Luo Ming Zhang
Department of Computer & Information Science & Engineering
University of Florida
{tali, sgchen, wluo, mzhang}@cise.ufl.edu

***Abstract*—Scan detection is one of the most important functions in intrusion detection systems. In order to keep up with the ever-higher line speed, recent research trend is to implement scan detection in fast but small SRAM. This leads to a difficult technical challenge because the amount of traffic to be monitored is huge but the on-die memory space for performing such a monitoring task is very limited. We propose an efficient scan detection scheme based on *dynamic bit sharing*, which incorporates probabilistic sampling and bit sharing for compact information storage. We design a maximum likelihood estimation method to extract per-source information from the shared bits in order to determine the scanners. Our new scheme ensures that the false positive/false negative ratios are bounded with high probability. 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 scan detection scheme reduces memory consumption by three to twenty times when comparing with the best existing work.**

I. Introduction

Many network-based attacks are preceded with a reconnaissance phase, in which the attacker or its zombies scan the hosts in a network to identify vulnerability. As a result, scan detection is one of the most fundamental functions in almost any network intrusion detection system (IDS). Cisco has been pushing for years to build security functions into its high-end routers. Scan detection is increasingly performed by routers with security modules or firewalls that inspect packets [1].

The throughput of high-speed links has been increased from 10 Mbps to 100 Mbps, multi-gigabits per second, and even terabits per second [2]. Modern firewalls heavily rely on ASIC chips to avoid becoming bottlenecks in the routing paths. In such a demanding environment, it is highly desirable to assist the fundamental IDS functions with hardware. Specifically, because scan detection requires storing a large amount of information extracted from the packets, DRAM can be too slow. Recent research suggests implementing this function in SRAM [3], [4], [5], [6].

On-die SRAM is fast but expensive. To achieve high speed, it is desirable to make on-die memory small — the access time for a SRAM of tens of kilobits is certainly much smaller than the access time for a SRAM of tens of megabits. Moreover, this high-speed memory has to be shared among other critical functions for routing, packet scheduling, traffic management and security purposes. The amount that will be allocated for online scan detection is likely to be a small fraction of the available SRAM. Therefore, it is extremely important to make scan detection, as well as other functions implemented in SRAM, memory-efficient.

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, 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 source is classified as a scanner if its spread exceeds a certain threshold. Therefore, scan detection is fundamentally an online traffic measurement problem.

One of the greatest challenges for scan detection is that the data volume to be stored can be huge. For example, the main gateway at our campus observes more than 10 million distinct source-destination pairs on an average day. Suppose each measurement period is one day long (in order to catch stealthy low-rate scanners). If we simply store all distinct source/destination address pairs for scan detection, it will require more than $80MB$ of SRAM, which is too much. A major thrust in the scan detection research is to reduce the memory consumption [3], [4], [6], [7], [8].

Reducing memory consumption does not come for free. The prior research sacrifices detection accuracy for memory saving. The basic idea is to compress the contact information in limited memory space. The compressed information allows us to estimate the spreads of the sources, instead of counting them exactly. However, the estimated spread values may cause false positives (in which a non-scanner is mistakenly reported as a scanner) and false negatives (in which a scanner is not reported). Consequently, the following questions become important for any practical security system: How serious is the false positive/false negative problem? Can the system be configured such that the false positive/false negative ratios are bounded? To date, few papers directly addressed these questions.

The prior work follows two general methods for memory reduction: *probabilistic sampling* and *storage sharing*. The probabilistic sampling method is to record only a certain percentage of randomly sampled contacts. An example is the one-level/two-level algorithms proposed by Venkataraman et al [3]. These algorithms store the source/destination addresses of the sampled contacts in hash tables. Their main contribution is to

derive the optimal sampling probability that ensures with high probability that the false positive/false negative ratios do not exceed certain pre-defined bounds.

However, it is not memory-efficient to directly store the addresses of the contacts made by each source. A naive solution is to use per-source counters to record the number of packets from each source. Near-optimal counter architectures such as counter braids [9] require only a few bits per source. The problem is that counters cannot remove duplicates: A thousand packets from the same source to the same destination should count as one contact, instead of a thousand. In order to remove duplicates, one may use Bloom filters [3] or bitmap algorithms [7]. They encode the contacts made by each source in a separate bitmap, which automatically filters duplicates. However, per-source bitmaps still take too much space. Cao et al. use a series of Bloom filters and a hash table to reduce the number of sources that need bitmaps [8].

Instead of using a separate bitmap for each source, an interesting space-saving method is to allow *storage sharing*, where each data structure is no longer dedicated to a single source but shared among multiple sources. This is particularly necessary when the number of sources is more than the number of available bits. Zhao et al. [6] encodes each contact in three shared bitmaps using a technique similar to Bloom filters. Yoon et al. [4] design another storage sharing method with superior performance. Although both methods can be used for scan detection, none of them provides any means to ensure that the false positive/false negative ratios are bounded. Moreover, our experiments show that these existing methods [8], [6], [4] take far more memory than the one proposed in this paper.

Also related is the work by Bandi et al. [10] using TCAM. Another research branch [10], [11], [12], [13], [14], [15], [16], [17], [18] is to find *heavy hitters*, i.e., sources that send a lot of packets. A heavy hitter may have a spread value of just one because it may send all its packets to the same destination. Hence, heavy hitters and scanners are very different.

This paper proposes an efficient scan detection scheme based on a new storage sharing method, called *dynamic bit sharing*, which shares the available bits uniformly at random among all sources, such that the memory space is fully utilized for storing contact information. It employs a maximum likelihood estimation method to extract per-source information from the shared bits in order to determine the scanners. It also enhances security through a private key. Our new method ensures that the false positive/false negative ratios are bounded. Moreover, given an arbitrary set of bounds, we show analytically how to choose the optimal system parameters such that the amount of memory needed to satisfy the bounds is minimized. We also perform experiments based on a real traffic trace and demonstrate that, using these optimal parameters, we can reduce the memory consumption by three to twenty times when comparing with the best existing work.

II. Problem Statement

The number of distinct destination addresses that an external source has contacted is called the *spread* of the source. The problem of scan detection is to configure a firewall or an intrusion detection system to report all external sources whose spreads exceed a certain threshold during a measurement period. We refer to these sources as *potential scanners* (or scanners for short).

If a firewall or an IDS keeps the exact count of distinct destinations that each source has contacted, it is able to report the scanners precisely. However, keeping track of per-source information consumes a large amount of resources. The limited SRAM may only allow us to estimate a rough count of distinct destinations that each source contacts [3], [4], [6]. When precisely reporting scanners is infeasible, the function of scan detection must be defined in a probabilistic term.

We adopt the *probabilistic performance objective* from [3]. Let h and l be two positive integers, $h > l$. Let α and β be two probability values, $0 < \alpha < 1$ and $0 < \beta < 1$. 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. Let k be the spread of an arbitrary source src. The objective can be expressed in terms of conditional probabilities:

$$\begin{aligned} &Prob\{\text{report } src \text{ as a scanner} \mid k \geq h\} \geq \alpha \\ &Prob\{\text{report } src \text{ as a scanner} \mid k \leq l\} \leq \beta \end{aligned} \tag{1}$$

We treat the report of a source whose spread is l or smaller as a *false positive*, and the non-report 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.

The memory requirement for detecting aggressive scanners is likely to be small. For example, suppose an aggressive scanner makes 100 distinct contacts each second, whereas a normal host rarely makes 100 distinct contacts in a day. To detect such a scanner, a firewall can set the measurement period to be a second. The number of contacts that pass the firewall in such a small period is likely to be small. Consequently, it does not need much memory to store them. However, the situation is totally different for stealthy scanners that make contacts at low rates. Consider a scanner that makes 500 distinct contacts a day. If the measurement period is a day, we are able to set it apart from the normal hosts. However, if the measurement period is a second, we will not detect this scanner because it makes less than 0.006 contact per second on average.

In order to detect different types of scanners, a firewall may execute multiple instances of a scan detection function simultaneously, each having a different measurement period. For aggressive scanners, a small period will be chosen so that they can be detected in real time. For stealthy scanners, a large period will be chosen. In the latter case, timely detection is of second priority because the scanners themselves operate slowly.

But the memory requirement is of first priority due to the large number of contacts that are expected to pass through the firewall in a long measurement period. Reducing memory consumption is the focus of this paper.

III. An Efficient Scan Detection Scheme

This section presents our efficient scan detection scheme (ESD).

A. Probabilistic Sampling

To save resources, a firewall (or IDS) samples the contacts made by external sources to internal destinations, and it only stores the sampled contacts. The firewall selects contacts for storage uniformly at random with a sampling probability p. The sampling procedure is simple: the firewall 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.

B. Bit-Sharing Storage

A bit array (also called *bitmap*) may be used to store all sampled contacts made by a source [7]. 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, we can estimate the number of contacts, i.e., the spread of the source, based on the number of zeros remaining in the bitmap. Using per-source bitmaps is not memory-efficient. On one hand, the size of each bitmap has to be large enough to ensure the accuracy in estimating the spread values of the scanners. On the other hand, the vast majority of normal sources have small spread values and their bitmaps are largely wasted because most bits remain zeros. To solve this problem, we want to put those wasted bits in good use by allowing bitmaps to share their bits.

To fully share the available bits, ESD stores contacts from different sources in a single bit array B. Let m be the number of bits in B. For an arbitrary source src, we use a hash function to pseudo-randomly select a number of bits from B to store the contacts made by src. The indices of the selected bits are $H(src \oplus R[0])$, $H(src \oplus R[1])$, ..., $H(src \oplus R[s-1])$, where $H(...)$ 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 each new source src' whose contacts are sampled by the firewall will be allocated a logical bitmap $LB(src')$ 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 firewall 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 second hash, $H(dst \oplus K)$, ensures that the bit is pseudo-randomly selected from $LB(src)$. The private key K is introduced to prevent the *hash collision attacks*. In such an attack, a scanner src finds a set of destination addresses, dst_1, dst_2, ..., that have the same hash value, $H(dst_1) = H(dst_2) = ...$ If it only contacts these destinations, the same bit in $LB(src)$ will be set, which allows the scanner to stay undetected. This type of attacks 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 reason, then a private key becomes necessary. Without knowing the key, the scanners will not be able to predict which destination addresses produce the same hash value.

To store a contact, ESD only sets a single bit and performs two hash operations. This is more efficient than the methods that use hash tables [3] or have features similar to Bloom filters that require setting multiple bits for storing each contact [6].

C. Maximum Likelihood Estimation and Scanner Report

At the end of the measurement period, ESD 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, ESD reports the source as a potential scanner. We will discuss how to keep track of the source addresses in Section III-D, and explain how to determine the threshold T in Section IV. Below we derive the formula for $\hat{k}$.

Let k be the true spread of source src, and n be the number of distinct contacts made by all sources. Let V_m be the fraction of bits in B whose values are zeros at the end of the measurement period, V_s be the fraction of bits in $LB(src)$ whose values are zeros, 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, V_m (or V_s, U_s) 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

$$q(k) = (1 - \frac{p}{m})^{n-k}(1 - \frac{p}{s})^{k}. \quad (2)$$

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:

$$L = q(k)^{U_s}(1 - q(k))^{s-U_s}. \quad (3)$$

In the standard process of maximum likelihood estimation, the unknown value k is technically treated as a variable in (3). We want to find an estimate $\hat{k}$ that maximizes the likelihood function. Namely,

$$\hat{k} = \arg\max_{k}\{L\}. \tag{4}$$

Since the maxima is not affected by monotone transformations, we use logarithm to turn the right side of (3) from product to summation:

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

From (2), the above equation can be written as

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

To find the maxima, we differentiate both sides:

$$\frac{\partial \ln(L)}{\partial k} = \ln(\frac{1-\frac{p}{s}}{1-\frac{p}{m}}) \cdot \frac{U_s - s(1-\frac{p}{m})^{n-k}(1-\frac{p}{s})^k}{1-(1-\frac{p}{m})^{n-k}(1-\frac{p}{s})^k}. \tag{5}$$

We then let the right side be zero. That is,

$$U_s = s(1-\frac{p}{m})^{n-k}(1-\frac{p}{s})^k. \tag{6}$$

Taking logarithm on both sides, we have

$$\begin{aligned}\ln\frac{U_s}{s} &= n\ln(1-\frac{p}{m}) + k(\ln(1-\frac{p}{s}) - \ln(1-\frac{p}{m})), \\ k &= \frac{\ln V_s - n\ln(1-\frac{p}{m})}{\ln(1-\frac{p}{s}) - \ln(1-\frac{p}{m})}.\end{aligned} \tag{7}$$

where $V_s = \frac{U_s}{s}$. 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

$$E(V_m) = (1-\frac{p}{m})^n. \tag{8}$$

Applying (8) to (7), we have

$$k = \frac{\ln V_s - \ln E(V_m)}{\ln(1-\frac{p}{s}) - \ln(1-\frac{p}{m})}. \tag{9}$$

Replacing $E(V_m)$ by the instance value V_m, we have the following estimation for k.

$$\hat{k} = \frac{\ln V_s - \ln V_m}{\ln(1-\frac{p}{s}) - \ln(1-\frac{p}{m})}, \tag{10}$$

where 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 pre-set parameters of ESD (see the next section).

D. Source Addresses

ESD does not store 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'. Hence, the frequency of storing source addresses is much smaller than the frequency at which contacts are sampled for setting bits in B. 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'). Second, a source may send thousands or even millions of packets through a firewall, 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). 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 develop the constraints that the system parameters must satisfy in order to achieve the probabilistic performance objective. Based on the 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 ESD to achieve the performance objective. Recall that on-die SRAM may be shared by other functions.

A. Report Probability

Consider an arbitrary source src whose spread is k. Given a set of system parameters, m, s, p and T, we derive the probability for ESD to report src as a scanner, i.e., $Prob\{\hat{k} \geq T\}$. From (10), we know that the following inequalities are equivalent.

$$\begin{aligned}\hat{k} &\geq T \\ \frac{\ln V_s - \ln V_m}{\ln(1-\frac{p}{s}) - \ln(1-\frac{p}{m})} &\geq T \\ V_s &\leq V_m(\frac{1-\frac{p}{s}}{1-\frac{p}{m}})^T\end{aligned}$$

Let U_s be the random variable for the number of '0' bits in $LB(src)$. $U_s = s \cdot V_s$. The above inequality becomes

$$U_s \leq s \cdot V_m \cdot (\frac{1-\frac{p}{s}}{1-\frac{p}{m}})^T. \tag{11}$$

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

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

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

U_s follows the binomial distribution with parameters s and $q(k)$, where $q(k)$ in (2) is the probability for an arbitrary bit in $LB(src)$ to remain zero at the end of the measurement period.

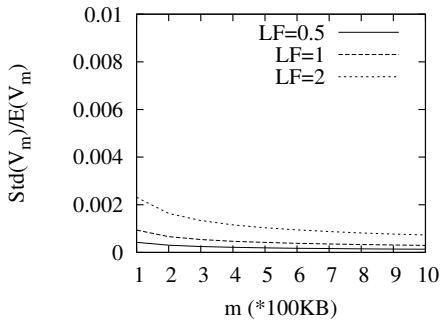

Fig. 1. The relative standard deviation, $\frac{Std(V_m)}{E(V_m)}$, approaches to zero as m increases. The *load factor* (LF) is defined as $n \cdot p / m$, where $n \cdot p$ is the number of distinct contacts that are sampled by ESD for storage. In our experiments (reported in Section V), when we use the system parameters determined by the algorithm proposed in this section, the load factor never exceeds 2.

Hence, the probability of having exactly i zeros in $LB(src)$ is given by the following probability mass function:

$$Prob\{U_s = i\} = \binom{s}{i} \cdot q(k)^i \cdot (1 - q(k))^{s-i}. \quad (12)$$

We must have

$$\begin{aligned} Prob\{\hat{k} \geq T\} &= Prob\{U_s \leq C\} \\ &= \sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(k)^i \cdot (1 - q(k))^{s-i}. \end{aligned} \quad (13)$$

B. Constraints for the System Parameters

We derive the constraints that the system parameters must satisfy in order to achieve the performance objective in (1). First, we give the variance of V_m, which is derived in Appendix A.

$$Var(V_m) \simeq \frac{e^{-\frac{np}{m}}(1 - (1 + \frac{np^2}{m})e^{-\frac{np}{m}})}{m}. \quad (14)$$

It approaches to zero as m increases. In Figure 1, we plot the ratio of the standard deviation $Std(V_m) = \sqrt{Var(V_m)}$ to $E(V_m)$, which can be found in (8). The figure shows that $Std(V_m)/E(V_m)$ is very small when m is reasonably large. In this case, we can approximately treat V_m as a constant.

$$V_m \simeq E(V_m) \simeq (1 - \frac{p}{m})^n. \quad (15)$$

The probabilistic performance objective can be stated as two requirements. First, the probability for ESD to report a source with $k \geq h$ must be at least α. That is, $Prob\{\hat{k} \geq T\} \geq \alpha, \forall k \geq h$. From (13), this requirement can be written as the following inequality:

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

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:

$$\sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(h)^i \cdot (1 - q(h))^{s-i} \geq \alpha. \quad (16)$$

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

$$\sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(l)^i \cdot (1 - q(l))^{s-i} \leq \beta. \quad (17)$$

C. Optimal System Parameters

Our goal is to optimize the system parameters such that the memory requirement, m, is minimized under the constraints (16) and (17). The problem is formally defined as follows.

$$\begin{aligned} &\textit{Minimize} \quad m \quad (18) \\ &\textit{Subject to} \quad \sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(h)^i \cdot (1 - q(h))^{s-i} \geq \alpha, \\ &\qquad \sum_{i=0}^{\lfloor C \rfloor} \binom{s}{i} \cdot q(l)^i \cdot (1 - q(l))^{s-i} \leq \beta, \\ &\qquad C = s \cdot (1 - \frac{p}{m})^n \cdot (\frac{1-\frac{p}{s}}{1-\frac{p}{m}})^T. \end{aligned}$$

The parameters, h, l, α and β, are specified in the performance objective. The value of n is decided based on the history data in the past measurement periods. To be conservative, we take the the maximum number n^* of distinct contacts observed in a number of previous measurement periods. More specifically, (8) 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.

$$\hat{n} = -\frac{m}{p} \ln V_m \quad (19)$$

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

$$Bias(\frac{\hat{n}}{n}) = E(\frac{\hat{n}}{n}) - 1 \simeq \frac{e^{\frac{np}{m}} - \frac{np^2}{m} - 1}{2np} \quad (20)$$

$$Std(\frac{\hat{n}}{n}) = \frac{\sqrt{m}}{np}(e^{\frac{np}{m}} - \frac{np^2}{m} - 1)^{1/2} \quad (21)$$

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^*.

To solve the constrained optimization problem (18), 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 (16) as a function $F_h(m, s, p, T)$, and

the left side of (17) 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 *non-increasing 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 $Potential(m, s, p)$, 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 $F_l(m, s, p, T) \leq \beta$ is satisfied. Because $F_h(m, s, p, T)$ is a non-increasing function in T, we need to find the smallest value of T that satisfies $F_l(m, s, p, T) \leq \beta$. That can be done numerically through binary search: Pick a small integer T_1 such that $F_l(m, s, p, T_1) \geq \beta$ and a large integer T_2 such that $F_l(m, s, p, T_2) \leq \beta$. 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, $F_l(m, s, p, T_1) \geq \beta$ and $F_l(m, s, p, T_2) \leq \beta$. The process stops when $T_1 = T_2$, which is denoted as T^*. The procedure $Potential(m, s, p)$ returns $F_h(m, s, p, T^*)$. The pseudo code is presented in Algorithm 1. We will omit the pseudo code for the other three procedures to save space.

Algorithm 1 $Potential(m, s, p)$

INPUT: m, s, p and β
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 while
$T^* = \bar{T}$;
return $F_h(m, s, p, T^*)$;

Essentially, what $Potential(m, s, p)$ returns is the maximum value of the left side in (16) under the condition that (17) is satisfied. The difference between $Potential(m, s, p)$ and α provides us with a quantitative indication on how conservative or aggressive we have chosen the value of m. If $Potential(m, s, p) - \alpha$ 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 $Potential(m, s, p) - \alpha$ 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 $Potential(m, s, p)$.

- Second, given a value of m and a value of s, we construct a procedure $OptimalP(m, s)$ that determines the optimal value p^* such that $Potential(m, s, p^*)$ is maximized. When the values of m and s are fixed, $Potential(m, s, p)$ becomes a function of p. It is a curve as illustrated in Figure 2; see explanation under the caption (a) and ignore the arrows in the figure for now.

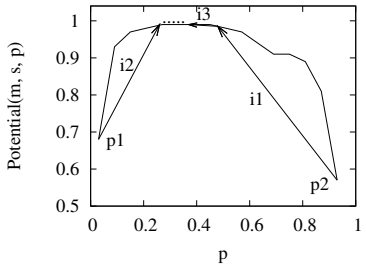

Fig. 2. **(A)** The curve (without the arrows) shows the value of $Potential(m, s, p)$ with respect to p when $m = 0.45MB$ and $s = 150$. Its non-smooth appearance is due to $\lfloor C \rfloor$ in the formula of $F_h(m, s, p, T^*)$. $F_h(m, s, p, T^*)$ depends on the values of $\lfloor C \rfloor$ and $q(h)$, which are both functions of p. **(B)** The arrows illustrate the operation of $OptimalP(m, s)$. 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 $(p_1 + p_2)/2$.

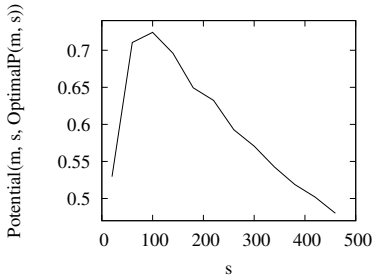

Fig. 3. The value of $Potential(m, s, OptimalP(m, s))$ with respect to s when $m = 0.25MB$.

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 $Potential(m, s, \bar{p}) < Potential(m, s, \bar{p} + \delta)$, 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 $OptimalP(m, s)$ 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 $OptimalP(m, s)$ is illustrated by the arrows in Figure 2; see explanation under the caption (b).

- Third, given a value of m, we construct a procedure $OptimalS\ (m)$ that determines the optimal value s^* such that $Potential(\ m, s^*, OptimalP(m, s^*))$ is maximized. When the value of m is fixed, $Potential(m, s, OptimalP(m, s))$ becomes a function of s. It is a curve as illustrated in Figure 3. We can use a binary search algorithm similar to that of $OptimalP(m, s)$ to find s^*.

- Fourth, we construct a procedure $OptimalM()$ that determines the minimum memory requirement m^* through binary search: Denote $Potential(m, OptimalS(m), OptimalP(m,$

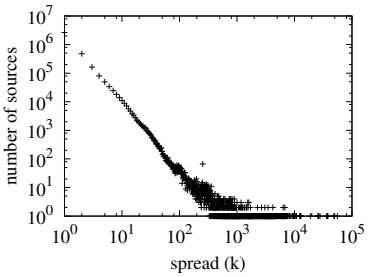

Fig. 4. Traffic distribution: each point shows the number of sources having a certain spread value.

$OptimalS(m)))$ as $Potential(m, ...)$. Pick a small value m_1 such that $Potential(m_1, ...) \leq \alpha$, which means that the performance objective is not met — more specifically, according to the semantics of $Potential(...)$, the constraint (16) cannot be satisfied if the constraint (17) is satisfied. Pick a large value m_2 such that $Potential(m_2, ...) \geq \alpha$, which means that the performance objective is met. Repeat the following operation. Let $\bar{m} = \lfloor (m_1 + m_2)/2 \rfloor$. If $Potential(\bar{m}, ...) \leq \alpha$, 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 $OptimalM()$.

In practice, a network administrator will first define the performance objective that is specified by α, β, h and l. He or she sets the value of n^* based on history data, and then sets $m = OptimalM()$, $s = OptimalS(m)$, $p = OptimalP(m, s)$ and T as the threshold value T^* before the last call to $Potential(m, s, p)$ is returned during the execution of $OptimalM()$. After the firewall (or IDS) is configured with these parameters and begins to measure the network traffic, it also monitors the value of n^*. If the maximum number of distinct contacts in a measurement period changes significantly, the values of m, s, p and T will be recomputed.

V. Experiments

A. Experimental Setup

We evaluate the performance of ESD and compare it with the existing work, including the Two-level Filtering Algorithm (TFA) [3], the Thresholded Bitmap Algorithm (TBA) [8], and the Compact Spread Estimator (CSE) [4]. 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 probabilistic performance objective in (1). TBA is not designed for meeting the probabilistic performance objective. It cannot ensure that the false positive/false negative ratios are bounded. CSE is designed to estimate the spreads of the external sources in a very compact memory space. It can be used for scan detection 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) [6] 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 [4]. 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, ESD performs two hash functions and makes one memory access.

The experiments use a real Internet traffic trace 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. Figure 4 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 ESD, TFA, TBA and CSE, and execute them with the traffic trace as input. As part of the setup in each experiment, the values of h and l are given to specify what to report as scanners. For example, if $h = 500$ and $l = 0.7h$, the sources whose spreads are 500 or more should be reported, and the sources whose spreads are 350 or less should not be reported. 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. The measurement period is one day. A long measurement period helps to separate low-rate scanners from normal hosts. 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 scan detection scheme to meet a given probability performance objective. Remarkably, the number of bits required by ESD is far smaller than the number of distinct sources in the traffic trace. That is, ESD requires much less than 1 bit per source to perform scan detection. 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 ESD and TFA for the amount of memory that they need in order to satisfy a given probabilistic performance objective, which is specified by four parameters, α, β, h, and l. See Section II for the formal definition of the performance objective. We do not compare TBA and CSE here because they are not designed to meet this objective.

The memory required by ESD is determined based on the iterative algorithm in Section IV-C. The values of other parameters, s, T and p, are decided by the same algorithm. Using these parameters, we perform experiments on ESD with the traffic trace as input, and the experimental results confirm that the performance objective is indeed achieved for each day during the week. The amount of memory required by TFA is determined experimentally based on the method in [3] together with the traffic trace. The parameters of TFA are chosen based on the original paper.

The memory requirements of ESD and TFA are presented in Tables I-II with respect to α, β, h and l. For $\alpha = 0.9$ and

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

	$l = 0.1h$			$l = 0.3h$			$l = 0.5h$			$l = 0.7h$		
h	ESD	TFA	ESD-1	ESD	TFA	ESD-1	ESD	TFA	ESD-1	ESD	TFA	ESD-1
500	0.09	2.02	0.33	0.19	2.53	0.43	0.30	3.61	0.54	0.97	6.12	1.01
1000	0.07	1.10	0.27	0.09	1.29	0.33	0.15	1.85	0.42	0.47	3.11	0.86
2000	0.03	0.55	0.24	0.05	0.71	0.29	0.08	1.02	0.42	0.25	1.62	0.86
3000	0.02	0.42	0.24	0.03	0.51	0.27	0.06	0.68	0.42	0.17	1.09	0.86
4000	0.01	0.32	0.21	0.03	0.38	0.27	0.03	0.52	0.42	0.13	0.83	0.86
5000	0.01	0.24	0.21	0.02	0.31	0.27	0.03	0.43	0.42	0.11	0.66	0.86

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

	$l = 0.1h$			$l = 0.3h$			$l = 0.5h$			$l = 0.7h$		
h	ESD	TFA	ESD-1	ESD	TFA	ESD-1	ESD	TFA	ESD-1	ESD	TFA	ESD-1
500	0.12	2.41	0.38	0.22	3.27	0.48	0.48	4.59	0.68	1.56	8.03	1.60
1000	0.08	1.29	0.32	0.12	1.65	0.38	0.24	2.34	0.50	0.76	4.04	1.20
2000	0.03	0.69	0.26	0.08	0.87	0.32	0.13	1.21	0.47	0.38	2.12	1.20
3000	0.02	0.46	0.26	0.06	0.60	0.32	0.09	0.83	0.47	0.26	1.42	1.20
4000	0.02	0.37	0.23	0.04	0.45	0.32	0.06	0.63	0.47	0.20	1.08	1.20
5000	0.01	0.29	0.23	0.04	0.35	0.32	0.05	0.52	0.47	0.16	0.89	1.20

$\beta = 0.1$, Table I shows that TFA requires six to twenty-four times of the memory that ESD requires, depending on the values of h and l (which the system administrator will select based on the organization's security policy). For example, when $h = 500$ and $l = 0.5h$, ESD reduces the memory consumption by an order of magnitude when comparing with TFA.

To demonstrate the impact of probabilistic sampling, the table also includes the memory requirement of ESD when sampling is turned off (by setting $p = 1$). This version of ESD is denoted as ESD-1. Since p is set as a constant, the iterative algorithm in Section IV-C needs to be slightly modified: The procedure $OptimalP(m, s)$ 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 $h = 5,000$ and $l = 0.3h$, ESD with sampling uses less than one thirteenth of the memory that is needed by ESD-1. However, when h becomes smaller or $\frac{l}{h}$ becomes larger, ESD 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 $l = 0.5h$, ESD with sampling uses 55.6% of the memory that is needed by ESD-1.

Table II compares the memory requirements when $\alpha = 0.95$ and $\beta = 0.05$. It shows similar results: (1) ESD uses significantly less memory than TFA, and (2) the probabilistic sampling method in ESD is critical for memory saving especially when h is large or $\frac{l}{h}$ is small. The table also demonstrates that the memory requirement of either ESD or TFA increases when the performance 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 [5], the authors also indicate that Bloom Filters [19], [20] 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-II, memory saving by ESD will still be significant.

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-scanners (whose spreads are no more than l) that are mistakenly reported as scanners. The *false negative ratio* (FNR) is the fraction of all scanners (whose spreads are no less than h) that are not reported by the system. In the previous subsection, we have shown that, given the bounds of FPR and FNR, it takes ESD 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 ESD with 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 $OptimalS(m, s)$ in Section IV-C to determine the value of s in ESD, use $OptimalP(m,\ s)$ 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. For $m = 0.05MB, l = 0.5h$, the results are presented in Tables III. We also perform the same experiments for CSE and TBA, and the results are presented in the table as well. The optimal parameters are chosen for each scheme based on the original papers.

When the available memory is very small, such as $0.05MB$ in Table III, CSE has zero FNR but its FPR is 1.0, which means it reports all non-scanners. 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

TABLE III
FALSE NEGATIVE RATIO AND FALSE POSITIVE RATIO OF ESD, CSE AND TBA WITH $m = 0.05MB$.

h	FNR			FPR		
	ESD	CSE	TBA	ESD	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

$h = 500$, its FNR is 26%. Only ESD 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 $h = 1,000$, they are 1.0% and 0.55%, respectively, while the FNR of TBA remains to be 26%.

VI. CONCLUSIONS

Scan detection 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 scan detection scheme based on a new method called *dynamic bit sharing*, 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 probabilistic performance 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.

REFERENCES

[1] R. Deal, "Cisco Router Firewall Security," *Cisco Press, ISBN-10: 1-58705-175-3*, August 2004.

[2] W. David Gardner, "Researchers Transmit Optical Data At 16.4 Tbps," *InformationWeek*, February 2008.

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

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

[5] Q. Zhao, A. Kumar, and J. Xu, "Joint Data Streaming and Sampling Techniques for Accurate Identification of Super Sources/Destinations," *Proc. of USENIX/ACM Internet Measurement Conference*, October 2005.

[6] Q. Zhao, J. Xu, and A. Kumar, "Detection of Super Sources and Destinations in High-Speed Networks: Algorithms, Analysis and Evaluation," *IEEE Journal on Selected Areas in Communications (JASC)*, vol. 24, no. 10, pp. 1840–1852, October 2006.

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

[8] J. Cao, Y. Jin, A. Chen, T. Bu, and Z. Zhang, "Identifying High Cardinality Internet Hosts," *Proc. of IEEE INFOCOM*, April 2009.

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

[10] N. Bandi, D. Agrawal, and A. Abbadi, "Fast Algorithms for Heavy Distinct Hitters using Associative Memories," *Proc. of IEEE International Conference on Distributed Computing Systems(ICDCS)*, June 2007.

[11] M. Charikar, K. Chen, and M. Farach-Colton, "Finding Frequent Items in Data Streams," *Proc. of International Colloquium on Automata, Languages, and Programming (ICALP)*, July 2002.

[12] G. Cormode and S. Muthukrishnan, "Space Efficient Mining of Multigraph Streams," *Proc. of ACM PODS*, June 2005.

[13] E. Demaine, A. Lopez-Ortiz, and J. Ian-Munro, "Frequency Estimation of Internet Packet Streams with Limited Space," *Proc. of Annual European Symposium on Algorithms (ESA)*, September 2002.

[14] X. Dimitropoulos, P. Hurley, and A. Kind, "Probabilistic Lossy Counting: An Efficient Algorithm for Finding Heavy Hitters," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 1, January 2008.

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

[16] P. Gibbons and Y. Matias, "New Sampling-based Summary Statistics for Improving Approximate Query Answers," *Proc. of ACM SIGMOD*, June 1998.

[17] G. Manku and R. Motwani, "Approximate Frequency Counts over Data Streams," *Proc. of VLDB*, August 2002.

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

[19] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[20] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Mathematics*, vol. 1, no. 4, June 2002.

APPENDIX A: 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,

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

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

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

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

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

Based on (8) and the equation above, we have

$$\begin{aligned} Var(V_m) &= E(V_m^2) - E(V_m)^2 \\ &= \frac{1}{m}(1 - \frac{p}{m})^n + \frac{m-1}{m}(1 - \frac{2p}{m})^n - (1 - \frac{p}{m})^{2n} \\ &\simeq \frac{e^{-\frac{np}{m}}(1 - (1 + \frac{np^2}{m})e^{-\frac{np}{m}})}{m}. \end{aligned} \tag{22}$$