Real-Time Detection of Invisible Spreaders

MyungKeun Yoon Shigang Chen
Department of Computer & Information Science & Engineering
University of Florida, Gainesville, FL 32611, USA
{myoon, sgchen}@cise.ufl.edu

***Abstract*—Detecting spreaders can help an intrusion detection system identify potential attackers. The existing work can only detect aggressive spreaders that scan a large number of distinct addresses in a short period of time. However, stealthy spreaders may perform scanning deliberately at a low rate. We observe that these spreaders can easily evade the detection because their small traffic footprint will be covered by the large amount of background normal traffic that frequently flushes any spreader information out of the intrusion detection system's memory. We propose a new streaming scheme to detect stealthy spreaders that are invisible to the current systems. The new scheme stores information about normal traffic within a limited portion of the allocated memory, so that it will not interfere with spreaders' information stored elsewhere in the memory. The proposed scheme is light weight; it can detect invisible spreaders in high-speed networks while residing in SRAM. Through experiments using real Internet traffic traces, we demonstrate that our new scheme detects invisible spreaders efficiently while keeping both false-positives (normal sources misclassified as spreaders) and false-negatives (spreaders misclassified as normal sources) to low level.**

***Index Terms*—stealthy spreader, network security, intrusion detection, streaming algorithm.**

I. Introduction

Monitoring and analyzing network logs is at the heart of identifying attackers in the early stage [1]. These logs can be low-level packet trace generated from routers or high-level audit records from network/host intrusion detection systems. In high-speed networks, such logs can come in large volume. To process them in real time, a fast and lightweight streaming algorithm is required, which should be able to work with limited memory and contiguously process incoming logs.

This paper studies the problem of detecting spreaders based on incoming logs that are tuples of source/destination addresses. We call an external source address a *spreader* if it connects to more than a threshold number of distinct internal destination addresses during a period of time (such as a day). We define the *cardinality* of a source to be the number of distinct destinations that the source have contacted. Similarly, we define the destination cardinality to be the number of distinct sources that have contacted the destination.

The reason for detecting spreaders is that attackers often begin with a reconnaissance phase of finding vulnerable systems before launching the actual attack. Suppose an attacker knows how to compromise a specific type of web servers. Its first step is to locate such web servers on the Internet. The attacker may probe TCP port 80 on all addresses in a target network by using Nmap [2]. To obtain more specific information, they may run an application-level vulnerability scanner such as Nessus [3] or Paros [4]. An intrusion detection system can inspect the incoming traffic and catch the reconnaissance packets, from which the spreaders are identified as potential attackers that demand extra attention.

It is not possible for a network security administrator to manually analyze the huge volume of logs produced by routers and intrusion detection systems in order to find spreaders. An automatic log-analyzing system is required. In fact, some intrusion detection systems have already implemented functions for identifying spreaders. For example, Snort [5] keeps track of the distinct destinations each source contacts in a recent period, and the length of the period is constrained by the amount of memory allocated to this function. The problem is that the existing systems are designed to catch "elephants" — aggressive spreaders whose cardinalities are so large that they easily stand out from the background of normal traffic. In response, a wily attacker will slow down the rate of its reconnaissance packets and let the normal traffic dilute the footprint of its activity. In the Snort case, the past records must be deleted to free memory once the allocated space is filled up by logs. If the attacker contacts a less-than-threshold number of destinations in each period during which logs of normal traffic will fill up the allocated space, it will stay undetected.

We note that even state-of-the-art intrusion detection systems can not detect stealthy spreaders if they send their packets at a low rate. These spreaders are called *invisible spreaders*. To catch them, we must make the detection system more sensitive. It is a cat-and-mouse game between attackers and defenders. As we build more and more sensitive detectors, the attackers will be forced to continuously reduce their reconnaissance rate in order to stay undetected. This will give more time for the network administrators to take action (such as patching systems) against the outbreak of new attacks. The attacks will become less effective if it will take them an exceedingly long time (e.g., months) to complete the reconnaissance phase over the Internet.

To design our new real-time detector for invisible spreaders, we observe (based on real Internet traffic traces) that normal traffic has strong skewness especially in an enterprise (or university campus) network. In particular, most inbound traffic is headed to a small number of servers for web, DNS, email, and business application services. Utilizing such skewness, we propose a new spreader detection scheme that is able to largely segregate the space used to store normal-traffic logs from the space used to store logs of potential spreaders. Due to such segregation, a large volume

978-1-4244-2324-8/08/$25.00 © 2008 IEEE.

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE "GLOBECOM" 2008 proceedings.

of normal-traffic logs will not cause the logs of spreaders to be flushed out of the memory. Furthermore, with a compact two-dimensional bit array based on Bloom filters, the new scheme can store a much larger amount of information about the spreaders, allowing previously invisible spreaders to be detected. We perform experiments based on real Internet traffic traces, and the results show that the proposed scheme is able to detect spreaders that are invisible to the existing detection systems and, at the mean time, keep both false positives (normal sources misclassified as spreader sources) and false negatives (spreader sources misclassified as normal sources) to low level.

The rest of the paper is organized as follows. Section II presents the design of our spreader detection scheme. Section III evaluates the proposed scheme via experiments using real Internet traffic traces. Section IV discusses the related work. Section V draws the conclusion.

II. Invisible-Spreader Detection

In this section, we propose a new scheme for detecting invisible spreaders. Our main technique is a novel streaming algorithm based on an *invisible-spreader detection filter*.

A. Invisible-Spreader Detection Filter (ISD)

Consider an intrusion detection system that is deployed to catch all external spreaders whose cardinality exceeds a threshold θ. In order to detect stealthy spreaders, one must set the value of θ small. However, as θ becomes smaller, the spreaders to be detected will have a smaller traffic footprint, and it is increasingly harder for the intrusion detection system to catch these spreaders without causing significant false-positive and false-negative ratios. When the small footprint of the stealthy spreaders is sufficiently diluted by normal traffic, the spreaders may even become *invisible* to the current detection system. To catch these invisible spreaders, more sensitive detection systems must be designed. Below we propose a new detector that can catch spreaders invisible to today's detectors (such as [6]).

Our *invisible-spreader detection filter* (ISD) uses an $n \times m$ bit array as its main data structure, which is initialized to be all zeros. Each bit $B(x, y)$ in the array is referenced by a row index x and a column index y. Bits will be set to ones to record the incoming connections made from external sources to internal destinations. A row is *empty* (or *non-empty*) if it has zero bit (or at least one bit) that is set to be one. There is a *row counter* $c(x)$ for each row x, storing the number of bits in the row that are set as one. The *fullness ratio R* of the filter is defined as $\frac{\sum c(x)}{n \times m}$, the percentage of bits in the array that are set to one. Similarly, the fullness ratio of row x is defined as $\frac{c(x)}{m}$, the percentage of bits in the row that are set to one. We define a *system parameter* α, specifying the desirable fullness. If $R > \alpha$, we reset the bit array to zeros.

Next we describe the operations of ISD. When receiving an input source/destination tuple (a, b), the filter computes k row indexes, $x_1 = h_1(a)$, ..., $x_k = h_k(a)$, and one column index, $y = h_{k+1}(b)$, where $h_1, ..., h_k$ are hash functions whose ranges are $[0..n-1]$ and h_{k+1} is a hash function whose range is $[0..m-1]$. The filter sets k bits, $B(x_1, y)$, ..., $B(x_k, y)$, to be one. Note that each column is actually a Bloom filter [7] [8]. The column index y selects a Bloom filter and the row indices specify the bits that together represent the tuple (a, b).

For each $i \in [1..k]$, if $B(x_i, y)$ was set from zero to one, the filter increases the row counter $c(x)$ by one. Rows indexed by x_1 through x_k are called the *representative rows* of source a in the filter. Bits $B(x_1, y)$ through $B(x_k, y)$ are called the *representative bits* of a. If the fullness of every representative row of source a is above a threshold β (whose value will be determined in the next subsection), ISD executes the following procedure to determine if a is a spreader.

1) For the jth column, let I_j be one if $B(x_i, j) = 1$ for all the representative rows of a. Otherwise, $I_j = 0$. We define
$$a_r = \sum_{j=0}^{m-1} I_j$$
2) The cardinality of a, denoted as $\hat{a_c}$, can be estimated based on the following formula given in [9].
$$\hat{a_c} = m \times ln(\frac{m}{m - a_r}) \quad (1)$$
3) If $\hat{a_c}$ is above θ, we consider source a to be a spreader.

Our column index, $y = h_{k+1}(b)$, is different from [6], which uses $y = h_{k+1}(a|b)$. This subtle yet critical difference helps ISD minimize the diluting effect of normal traffic over the small traffic footprint of invisible spreaders. Suppose a destination address b represents a busy webserver in an enterprise network, and millions of client users connect to b. If $y = h_{k+1}(a|b)$ is used, these clients will fill up the whole bit array with ones since the source addresses a randomizes the column index y. To the contrary, only one column of the bit array will be set to ones if $y = h_{k+1}(b)$ is used. Our Internet trace shows that the vast majority of normal traffic is directed to a small number of servers. Our scheme concentrates such normal traffic to a small number of columns in the bit array, leaving the rest of the array for detecting spreaders. Hash collisions may cause false positives. By tuning the system parameters, we can control the level of false positive, as well as the level of false negative.

B. Parameter Configuration

The goal of ISD is to identify spreaders whose cardinality values are larger than θ, which is given as a user requirement. Let M $(= n \times m)$ be the size of the allocated memory. The performance of ISD is affected by the selection of the following system parameters: α, β, m, and n. Below we discuss how to set these parameters.

1) We first set the values for β and m. According to the previous subsection, a spreader will be detected when the following condition is satisfied.
$$m \times ln(\frac{1}{1 - \frac{a_r}{m}}) \geq \theta. \quad (2)$$

TABLE I: Parameter Configuration Examples ($c = 10$)

θ	100	200	300	400	500	600	700	800	900
β	0.790	0.790	0.904	0.790	0.858	0.904	0.935	0.790	0.828
m	64	128	128	256	256	256	256	512	512
α	0.249	0.365	0.463	0.478	0.547	0.598	0.634	0.571	0.612

Based on their definitions, we can approximate β as $\frac{a_r}{m}$. Applying this approximation to (2), we have the following formula for setting the value of β and m.

$$\beta = 1 - e^{-\frac{\theta}{m}}. \quad (3)$$

Recall that the parameter β is used to trigger the procedure for determining a possible spreader. When the value of β is set by the above formula, once triggered the procedure is likely to find a spreader.

The problem is that there are two undecided parameters in the formula. We observe that small m is desirable for ISD. This is because small m allows large n, which reduces hash collisions among row indices. Consequently, large β is preferable. However, if β is very close to one, ISD may suffer from hash collisions among column indices. In this paper, we choose β to be below 0.95, but it can be adjusted according to any specific application or deployment environment. Once β is chosen, m can be set based on (2). Alternatively, we may also set m first and then calculate β from (2). For example, it is natural to choose m as a multiple of words, which makes it easy to fit the bit array in memory. For each m $(= 32, 64, ...)$, we compute β and choose the largest β below 0.95. Table I shows some examples for parameter configuration. It shows how β and m are determined for θ from 100 to 900. After m is determined, n is calculated as $\frac{M}{m}$.

2) We now determine the value of α. First we examine how α affects the detection of spreaders. When α is too larger, the bit array of ISD will be overly populated with ones, causing frequent hash collisions and resulting in false positives — a non-spreader is claimed as a spreader because its representative rows are populated with ones by tuples of other sources (due to hash collisions). If α is too small, false positives may hardly happen, but the filter will be frequently reset to zeros, losing the already-recorded information about spreaders and resulting in false negatives — failure in detecting spreaders. Next we will use some statistical properties to determine the value of α.

Suppose ISD only receives normal traffic for a period of time and its bit array is mostly set by the normal traffic. Let Y be a random variable that represents $c(x)$ for row x in the bit array. The expectation and the variance of Y are given below. We omit the derivation process due to page limit.

$$E(Y) = \alpha \times m \quad (4)$$

$$V(Y) = \alpha \times m \times (1 - \alpha) \quad (5)$$

From $E(Y)$ and $V(Y)$, we can define a statistical upper-bound for Y as follows:

$$U(Y) = E(Y) + c\sqrt{V(Y)} \quad (6)$$

where statistical error will be small if the constant c is large. Eq. (6) means that there is a high probability that $c(x)$ is below $U(Y)$ if x is a representative row for only normal traffic. On the contrary, if x is a representative row for any spreader, $c(x)$ should be larger than $U(Y)$. Hence, based on the above equations, we can set the value of α as follows.

$$\alpha = \frac{2\beta m + c^2}{2(m + c^2)} - \sqrt{(\frac{2\beta m + c^2}{2(m + c^2)})^2 - \frac{m\beta^2}{m + c^2}}. \quad (7)$$

Table I shows α as a result of the proposed heuristic method to configure β, m and α when $c = 10$.

III. Experiment

We evaluate ISD using real-world Internet traffic traces. We implemented not only ISD but also the advanced scheme from [6], which we call *online streaming module* (OSM). We compare their false positives and false negatives. The experimental results confirm that ISD detects invisible spreaders while minimizing the negative impact of normal traffic.

A. Traffic Trace and Implementation Details

In these experiments, we set k to 3. Large k is helpful to differentiate sources, but it increases processing time and fills up quickly the bit table. A good argument for $k = 3$ can be found in [6].

We use packet header traces gathered at the gateway routers of the University of Florida in the U.S. The trace was collected for 24 hours and we take only the inbound session from the Internet. It contains 751,286 distinct source IP addresses, 120,916 distinct destination IP addresses and 2,427,327 distinct source/destination tuples. Note that we denote the source IP address of a packet as a and the destination IP address as b in our notation of packet (a, b). In this sense, the goal of the experiment is to find heavy spreaders of horizontal network scans [10].

Figure 1(2) illustrates the traffic pattern with respect to source(destination) cardinality. The x-axis is the number of sources(destinations) whose cardinality lies between x and $2 \times x - 1$. Each figure has two graphs of cumulative ratios for the number of distinct sources(destinations) and the number of distinct source/destination tuples. In figure 1, we see that 86% of the total sources contact less than 4 distinct destinations and 99% of them contact less than 32 distinct destinations. Figure 1 shows that the number of source/destination tuples increases just as the number of sources does. Therefore, we can not see any skewness in the figure. However, we can see a different pattern in figure 2. Two curves seem different. The figure shows that only some of the destinations occupy most of the source/destination tuples. For example, at $x = 8$, the accumulated number of destinations is above 97%, but their aggregated source/destination tuples are below 27%. It means that less than top 3% servers occupy more than 73% of the total source/destination tuples. Exploiting

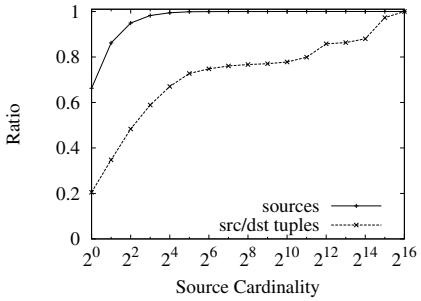

Fig. 1: Cumulative ratios of the numbers of distinct sources and distinct source/destination tuples with respect to source cardinality

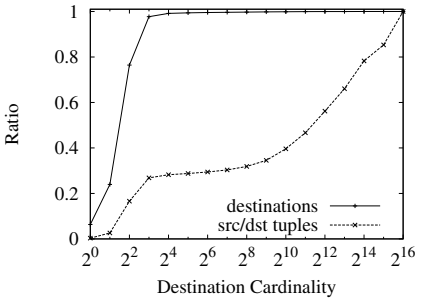

Fig. 2: Cumulative ratios of the numbers of distinct destinations and distinct source/destination tuples with respect to destination cardinality

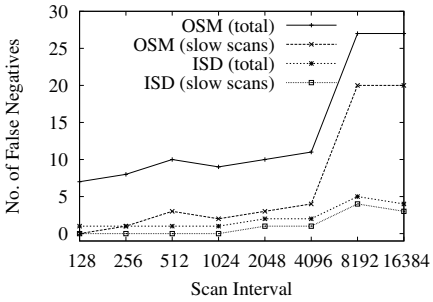

Fig. 3: Number of false negatives when M=256KB

this skewness, ISD has the edge on other intrusion detection systems.

For all the experiments, we set θ to be 500. It means that we take any source of cardinality above 500 as a spreader or scan source. With $\theta = 500$, the original traffic trace already includes 75 spreaders. We also generate some artificial scan packets to simulate invisible spreaders. For each experiment, we add 20 artificial slow scan sources to the original traffic trace. These source addresses are carefully chosen so that the original traffic trace does not include any same source address as the artificial scan sources. Each artificial scan source will send a total of λ distinct scan packets. It generates a scan packet every other μ normal source/destination tuples. The default parameters for the experiments are as follows: $M = 256KB, m = 256, n = 8,192, \alpha = 0.547, \alpha_O = 0.4, \theta = 500, \lambda = 600, \mu = 1,024, k = 3$. Note that β and α are determined by equations 3 and 7.

For comparison purpose, we also implemented OSM from [6]. For a fair comparison, both bit tables of OSM and ISD have the same memory size M. To optimize OSM, the maximum number of one-bits for OSM is set to α_O, which is different from α. Through the experiments, we observe that OSM degrades if α_O is set too large or too small. The default value of α_O is 0.4. Once the ratio of one-bits is above α_O, the decoding process runs and OSM restarts in a clean state.

For each experiment, we compare false negative(positive) sets of OSM and ISD. We use $FN_O(FN_R)$ to denote the false negative set of $OSM(ISD)$. Similarly, we use $FP_O(FP_R)$ to denote false positive sets. Let RS be a set of real spreaders, which has 95 sources (75 spreaders from the original traffic trace and 20 artificial scan sources). Let $D_O(D_R)$ be a set of detected sources by $OSM(ISD)$. We define FN_O, FN_R, FP_O and FP_R as follows: $FN_O = RS - D_O$, $FN_R = RS - D_R$, $FP_O = D_O - RS$, $FP_R = D_R - RS$.

B. Experimental Results

Figures 3∼6 compare the numbers of false negatives(positives) between ISD and OSM. The x-axis of each figure is μ, the number of normal source/destination tuples between two slow scan packets. A large value of μ implies that the attacker further slows down in sending the scan packets. In figure 3, we have four curves. OSM(total) is the number of false negatives of OSM, so it equals $|FN_O|$ with μ from 128 to 16,384. OSM(slow scans) is the number of false negatives, but we only count the artificial slow scan sources that are not detected. Therefore, its maximum value is 20 as we have 20 artificial slow scan sources. The same notations are used for ISD such as ISD(total) and ISD(slow scans). Note that ISD(total) plots $|FN_R|$.

Figure 3 shows that ISD catches most spreaders until μ becomes 4,096. Even when $\mu = 16,384$, ISD catches 17 artificial spreaders out of 20. To the contrary, OSM misses much more spreaders than OSM. Even when $\mu = 128$, it misses 7 non-artificial spreaders. It starts missing artificial scan sources at $\mu = 256$. At $\mu = 8,192$, OSM can not detect any slow scan sources while ISD detects 16 out of 20. Note that we trade false positives with false negatives when designing ISD, but false positives should be controlled by setting α to be tight. Figure 4 shows it. Even at $\mu = 16,384$, ISD only triggers 9 false positives. Considering that the number of source/destination tuples is above two millions, this false positives may be accepted in most applications.

We repeat the same experiment with different n. Figures 5 and 6 show the result with $n = 32,768$, which means $M = 1MB$. In this experiment, ISD does not miss any spreaders including slow scan sources except one at $\mu = 16,384$. Note that both ISD(total) and ISD(slow scans) remain zero until $\mu = 8,192$. To the contrary, OSM still misses some spreaders as shown in the figure. It can not detect 8 out of 20 slow scan sources at $\mu = 16,384$ even though the memory size has quadrupled. It is encouraging that ISD accomplishes better detection accuracy even when M is as small as $256KB$. Figure 6 shows that ISD triggers only small false positives.

IV. Related Work

Snort is a world famous network-based intrusion detection system. To detect scan sources, it simply keeps track of each source and the set of distinct destinations it contacts within a specified time-window. Thus, the memory require-

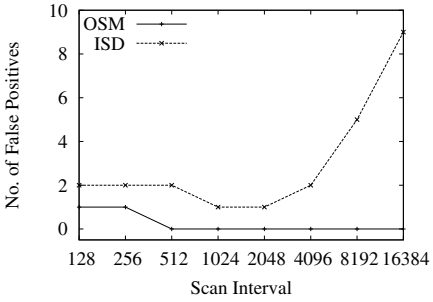

Fig. 4: Number of false positives when M=256KB

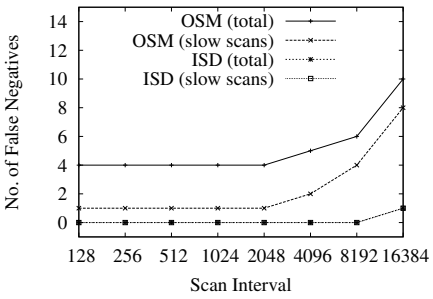

Fig. 5: Number of false negatives when M=1MB

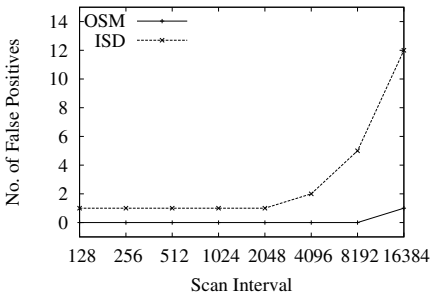

Fig. 6: Number of false positives when M=1MB

ment should be at least the total number of distinct source-destination pairs within the monitoring period, which is not feasible for detecting invisible spreaders [5], [11].

Venkataraman et al. define a heavy distinct-hitters problem [11] as follows: given a stream of (x, y) pairs, find all the x's that are paired with a large number of distinct y's. The heavy distinct-hitters problem can be used to define the scan detection problem. The authors also define a more specific problem, superspreaders, and propose two techniques to detect superspreaders. Superspreaders are heavy distinct-hitters, but they scan victims quickly. In other words, the monitoring period is short. Therefore, the proposed techniques can detect network scans only if they finish attacks within the short monitoring period.

Recently, an efficient streaming algorithm was proposed by Zhao et al. [6], which we call *Online Streaming Module* (OSM) in this paper. They approached the issue by devising a traffic measurement tool, which approximately estimates the cardinality of a source. It uses a two-dimensional bit table, which utilizes the memory space compactly. However, it suffers from the normal traffic volume as other $IDSes$ do. Our proposed scheme uses a two-dimensional bit table as the basic data structure to save memory space, but we use a new indexing to overcome the problem of huge normal traffic. It was not achieved by any previous work.

Recently, Gao et al. propose to detect stealthy spreaders by using online outdegree histograms in the context of change detection [12]. Their scheme is also based on two alternating time-windows. We emphasize that they use the definition of stealthy spreaders different from ours. In their definition, stealthy spreaders are a group of sources who send scanning packets at a constant rate together. Actually, they aim to detect scans from botnets. To the contrary, we focus on detecting heavy spreaders who may seem invisible in that they can evade detection thanks to the large amount of normal traffic.

All of the above techniques work fine for a short monitoring period. However, none of them detect invisible spreaders if the malicious sources send scanning packets slowly and steadily.

V. Conclusion

We defined the invisible spreader detection problem and showed the negative effects of normal traffic. To the best of our knowledge, none of the current $IDSes$ are free from these problems. We proposed a novel streaming algorithm to detect invisible spreaders and to mitigate the negative effects of normal traffic. By experiments on real-world Internet traffic traces, we confirmed the advantages of the proposed scheme. It is expected to help network/security management people in practice to detect general slow attacks, including invisible spreaders, which have been believed a difficult problem in network security.

References

[1] B. Schneier, "SIMS: Solution, or Part of the Problem?" *IEEE Security and Privacy*, vol. 2, no. 5, October 2004.
[2] "nmap," *http://www.insecure.org/nmap*, 2008.
[3] "nessus," *http://www.nessus.org/nessus/*, 2008.
[4] "paros," *http://www.parosproxy.org*, 2008.
[5] "Snort," *www.snort.org*, 2008.
[6] Q. Zhao, J. Xu, and A. Kumar, "Detection of Super Sources and Destinations in High-Speed Networks: Algorithms, Analysis and Evaluation," *IEEE JSAC*, vol. 24, no. 10, October 2006.
[7] 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.
[8] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Mathematics*, vol. 1, no. 4, June 2002.
[9] K. Hwang, B. Vander-Zanden, and H. Taylor, "A linear-time probabilistic counting algorithm for database applications," *ACM Transactions on Database Systems*, vol. 15, no. 2, June 1990.
[10] S. Staniford, J. Hoagland, and J. McAlerney, "Practical Automated Detection of Stealthy Portscans," *Journal of Computer Security*, vol. 10, pp. 105 – 136, 2002.
[11] S. Venkatataman, D. Song, P. Gibbons, and A. Blum, "New Streaming Algorithms for Fast Detection of Superspreaders," *Proc. of NDSS'05*, Feb. 2005.
[12] Y. Gao, Y. Zhao, R. Schweller, S. Venkataraman, Y. Chen, D. Song, and M. Kao, "Detecting Stealthy Spreaders Using Online Outdegree Histograms," *Proc. of IEEE International Workshop on Quality of Service'07*, pp. 145–153, June 2007.