Detecting Stealthy Spreaders Using Online Outdegree Histograms

Yan Gao, Yao Zhao, Robert Schweller, Shobha Venkataraman†,
Yan Chen, Dawn Song† and Ming-Yang Kao,
Northwestern University, Evanston, IL 60208, USA,
†Carnegie Mellon University, Pittsburgh, PA 15213, USA

ABSTRACT

We consider the problem of detecting the presence of a sufficiently large number of hosts that connect to more than a certain number of unique destinations within a given time window, over high-speed networks. We call such hosts *stealthy spreaders*. In practice, stealthy spreaders can be symptomatic of botnet scans or moderate worm propagation. Previous techniques have focused on detecting sources with an extremely large outdegree. However, such techniques will fail to detect spreaders such as bot scans in which each scanning host will scan only a moderate, fixed number of destinations. In contrast, our scheme maintains a small, fixed size memory usage, and is still able to detect stealthy spreader scenarios by approximating outdegree histograms from continuous traffic. To the best of our knowledge, we are the *first* to study the efficient outdegree histogram estimation and stealthy spreader detection problems. Evaluation based on real Internet traffic and botnet scan events show that our scheme is highly accurate and can operate online.

1. INTRODUCTION AND MOTIVATION

The ever-increasing link speeds and traffic volumes of the Internet make monitoring and analyzing network traffic a challenging but essential service for managing and securing ISPs and enterprise networks. The basic goal of network monitoring is to detect traffic changes, and indicate attacks, misbehavior, victims in terms of change patterns.

There are two highly desirable performance features for high-speed network monitoring: 1) a small amount of memory usage (to be implemented in SRAM); 2) a small number of memory accesses per packet [1, 2]. Many data streaming algorithms have been proposed to this end. For example, online entropy estimation approaches [3, 4] are introduced to detect if there exists unexpected changes in the network traffic and report alarms if an anomaly is found. However, entropy itself is an aggregated metric, which loses some important information that describes the distribution or histogram. In general, online histogram estimation, *e.g.*, algorithms [5, 6], can provide more information on the features of network traffic. However, these algorithms cannot record the number of *unique* items in the data stream. For instance, if a source has several parallel connections to the same destination, its histogram will have no difference from the one in which a spreader connects to multiple destinations in the recording period.

Recently, streaming schemes [7, 8] have been proposed to detect superspreaders, which are sources that connect to a large number of distinct destinations. This work is motivated by the observation that a compromised host doing fast scanning for worm propagation often makes an unusually high number of connections to distinct destinations within a short time.

Nowadays, botnets, each being an army of compromised hosts, dominate the Internet attack landscape. A botnet is controlled by a botmaster and usually scans at a much slower rate in order to evade detection. Bots engaging in such activities constitute *stealthy spreaders*. In general, stealthy spreaders are characterized by a number of sources sending a certain number of connection requests (often unsuccessful) to different destinations within a short time period. The number of scans by any single source is lower than the heavy hitter threshold, but a great number of such sources combined together can constitute a large scale attack. Stealthy spreaders can be responsible for moderate worm propagation, or botnet scanning. Therefore, early detection is extremely important. As analyzed in Section 4.3, existing superspreader schemes will be too expensive to detect such stealthy scans. In general, the superspreader detection problem is a special case of spreader detection. Stealthy spreader detection in particular poses a much greater challenge and currently no quality streaming methods exist for high speed networks.

In this paper, we study the problem of detecting stealthy spreaders with constant small memory and few memory accesses per packet, in real time. Our algorithms detect stealthy spreaders by approximating the outdegree histogram from continuous traffic. The outdegree histogram is a histogram of the number of hosts that connect to a certain number of distinct destinations. By using the outdegree histogram as the feature, we argue the change of approximate histograms observed in flow traces reveals the presence of anomalies, which may be caused by stealthy spreaders.

Our spreader detection system consists of two phases: the data recording phase and the spreader detection phase. For the former, we propose two algorithms: sampling and coupon collecting-based. For the latter, we use linear regression to estimate the average outdegree of a spreading event (*e.g.*, a botnet scan) and the number of unique sources for this event. To the best of our knowledge, we are the first to estimate the distribution of host outdegree online and use it for stealthy spreader detection.

We evaluate the detection accuracy and performance of

our approaches with real Internet traffic traces from an OC48 network collected by the Cooperative Association for Internet Data Analysis (CAIDA) and real botnet scan events observed in a class B network. Our experimental results show that when the scan flows constitute more than 2% of the total number of flows in the traffic, our detection algorithm can accurately identify the scan intensity as well as the average scanning outdegree of the scanners. As for the real scan event, we can accurately detect it with less than a 10% error on the scan outdegree and intensity when buried in most of the normal CAIDA traffic traces.

The outline for the rest of the paper is as follows. In Section 2 we dicuss related work. In Section 3 we present our problem formulation and the system architecture. In Section 4 we present our detection algorithms. In Section 5 we discuss the evaluation methodology and show the results. Finally, we conclude in Section 6.

2. RELATED WORK

Recently there has been an interest in using entropy and traffic distribution features for different networking measurements and intrusion detection applications [9, 3]. Computing the entropy of a distribution conceptually provides much greater sensitivity than the pure volume based methods. Chakrabarti *et al.* [3] proposed an algorithm to estimate "entropy norm" and entropy itself. Guha *et al.* [4] outlined algorithms estimating entropy and other information-theoretic measures in the streaming context, but they require knowledge of the underlying distribution. Lall *et al.* [9] used a similar algorithm as [3], but modified and extended it with the sieving approach. They provide unbiased estimates of the entropy, and do not make strong assumptions regarding the underlying distribution. However, entropy itself is an aggregated metric, which loses some concrete information that describes the distribution. Further, entropy does not record the number of unique items in the data stream.

Streaming algorithms have been used in the networking area to estimate the frequency moments and flow size distribution, and to identify heavy-hitters or heavy-distinct-hitters. For example, [7, 8] propose some good streaming schemes to detect superspreaders, which are sources or destinations that have communicated with a large number of distinct destinations or sources during a small time interval.

Work has also been done on the problem of maintaining approximate optimal online histograms [5, 6]. In particular, [6] provides provably efficient online algorithms. However, in this work the histograms considered are piecewise constant approximations of the function mapping keys (IP addresses) to traffic volume. As this type of histogram is very different from the type considered in this paper, the work is not directly applicable.

Our work makes use of previous distinct counting algorithms [10, 11, 12]. With distinct counting algorithms, for example, one can accurately count how many distinct sources appear in a data stream with very little memory. However, it is not a simple task to directly extend it to detect stealthy spreaders, which is the focus of our paper.

3. ONLINE OUTDEGREE HISTOGRAM DETECTION SYSTEM

In this section we introduce the problem definition of finding outdegree histograms as well as describe the high level architecture of the detection system.

3.1 Problem Definition

At a high level, we aim to solve two problems: (1) report an approximation of the outdegree histogram of the input data stream; (2) directly detect the presence of stealthy spreaders without necessarily reporting the complete outdegree histogram. Moreover, because of the large volume of network traffic, constant-space and single-pass algorithms are desirable. For both of these problems we design online single-pass algorithms that obtain information about the outdegree histogram of the data in constant space.

Formally, we describe our problem as follows. Let S be an input stream of source-destination pairs, and z be a positive integer whose exponential powers define the buckets of the histogram ($z = 2$ by default in our implementation). For bucket i, let W_i be the set of sources such that a source s is in W_i if and only if the number of unique destinations that s connects to is at least z^i and at most $z^{i+1} - 1$. Let $m_i = |W_i|$ and let q_r be the number of sources that connect to r distinct destinations. Then $m_i = \sum\limits_{r=z^i}^{z^{i+1}-1} q_r$. The problem of creating an approximate histogram is to estimate m_i for $0 \leq i \leq d$. We note that our algorithm can be easily adapted to other forms of dividing buckets, not just buckets of an exponential series. However, this division ensures that a small (logarithmic) number of buckets is sufficient to cover all ranges of outdegree.

In this paper, to accurately detect stealthy spreaders, we work in the context of change detection. In practice, network traffic changes constantly, making the outdegree histogram of the network traffic, *i.e.*, the absolute values of m_i, change as well. Therefore we may not able to tell whether stealthy scans occur simply from the outdegree histogram. Change detection is a common approach used to detect anomalous activities. We consider the change of the outdegree histogram of two time intervals for detecting a stealthy scan.

Let $m_{1,i}$ denote the number of distinct sources that connect to $n \in [z^i, z^{i+1} - 1]$ distinct destinations within the first of two temporally adjacent data streams. Similarly we define $m_{2,i}$ for the second stream. We are interested in the value $m_i' = m_{1,i} - m_{2,i}$.

In this paper, we mainly focus on the application of cooperative stealthy scan detection. Experimental evidence has shown that the outdegree of a cooperative stealthy scan typically follows a quasi-normal distribution (See Section 5.3.3). Thus, in practice a scan constitutes a large surge within a single bucket, with relatively little contribution in other buckets. Therefore, we assume that in the absence of a cooperative scan, the change histogram value m_i' should be below some small threshold c_l. In contrast, in the presence of an attack within bucket i, the change value m_i' should be large, above some threshold c_h. Our goal is to detect
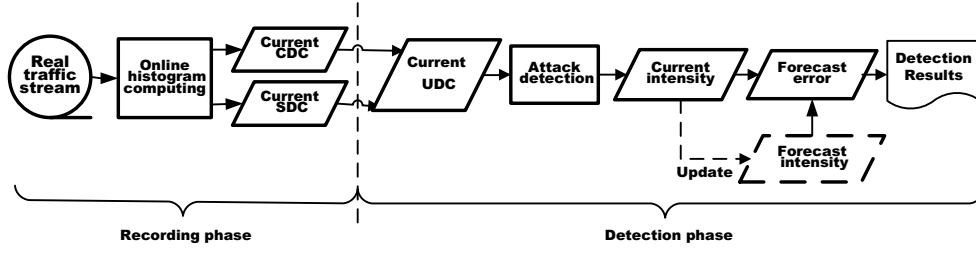
Figure 1: System architecture

when some m_i' exceeds c_h, while ignoring the normal case in which each m_i' is below c_l.

3.2 System Architecture

As the first step, we aim to detect classical TCP stealthy spreaders, such as worms and scans. We choose to build the histogram for unsuccessful connections as the basis for detection. In current network traffic, hot spots in P2P networks, web servers and game servers all exhibit similar characteristics as scanning in that one source IP may send out a large number of connections to different destination IPs. However, most of the connections of P2P super nodes, web servers or game servers are successful. On the contrary, (random) scanners usually have a large number of unsuccessful connections due to unused IPs or firewalls. Therefore, to avoid false alarms, we only consider unsuccessful connections. To deterministically determine if a connection is successful or not, one needs to store some state information while receiving a SYN packet and waiting for the SYN_ACK packet. This may consume a large volume of memory, even if sampling approaches are introduced. Therefore, we instead build two approximate histograms. One is used for all the connections (CHIS) (including both successful connections and unsuccessful connections). This is constructed by counting distinct SYN packets. The other is only used for successful connections (SHIS), *i.e.*, the connection includes a SYN_ACK packet as well as SYN packet. This is constructed by counting the distinct SYN_ACK packets. If there are no spreaders in the traffic, the histograms of these two data streams are close to each other. In contrast, anomalies will cause different changes in the histogram. For example, superspreaders will introduce changes in large buckets, while stealthy spreaders will make changes in middle buckets of the histogram.

The system as a whole consists of two key phases, the recording phase and the detection phase, as Figure 1 shows. In the recording phase, we propose two efficient algorithms, as mentioned in Section 4.1, to build two interim data structures online. One is the distinct counting structure for all the connections (CDC), the other is the distinct counting structure for successful connections (SDC). These two interim data structures can be used to construct the outdegree histogram for CHIS and SHIS respectively. Our basic scheme is to detect anomalous stealthy spreaders by measuring the distribution change of hosts' outdegree in real time. In practice, to reduce memory consumption and detection speed, we can alternatively use the interim data structure directly for detection without loss of accuracy. We show the relationship between the interim data structure and the outdegree histogram in Section 4.2.1.

Therefore, at the end of each time interval, we subtract SDC from CDC to form the distinct counting structure for unsuccessful connections (UDC). We then use our stealthy spreader detection scheme to get the estimation of outdegree and intensity (the product of the number of scanners and their outdegree) of the spreader. By considering the possible background noise of the traffic, a time series analysis method is applied to obtain the forecast change of the spreader intensity. If the error between the current spreader intensity and the forecast spreader intensity is larger than the threshold, and the outdegree is larger than our lower bound, there may exist stealthy spreaders, and vice versa.

4. DETECTION ALGORITHMS

In this section we describe our algorithms for the detection of cooperative stealthy spreaders. As the first step, we aim to detect some classical spreaders, such as TCP worm propagation and scans.

Our detection algorithm has two phases; the *recording phase* and the *spreader detection phase*. In the recording phase, the detector captures consecutive packets at a peering point in each interval and creates a detection data structure. Our algorithms for this phase are described in Section 4.1. In the spreader detection phase, we apply linear programming, linear regression, or other heuristics to extract approximate outdegree histogram information from the sampling data structure. Our spreader detection algorithms are described in Section 4.2. These include algorithms for reporting approximate histograms (Section 4.2.1) and for directly detecting stealthy spreaders (Section 4.2.2).

4.1 Recording Phase

In the recording phase our system creates a vector of values l_i using either a sampling algorithm or a coupon collecting based algorithm. From this vector, approximate histogram and spreader detection information can be obtained in the spreader detection phase. Both algorithms achieve a vector with the same expected value and thus can be used interchangeably in the spreader detection phase. Both techniques also use the same amount of memory.

However, there is a tradeoff between speed and accuracy. The sampling algorithm updates a smaller number of counters per packet in the data stream and is thus faster. However, the coupon collecting algorithm creates a vector of values whose variance from expectation is smaller, creating a better approximation interim structure. The description of the two algorithms is as follows. We note that in our experimentation we focus on the coupon collecting algorithm as the improvement in accuracy is substantial and the speed, while slower than sampling, is fast enough in practice.
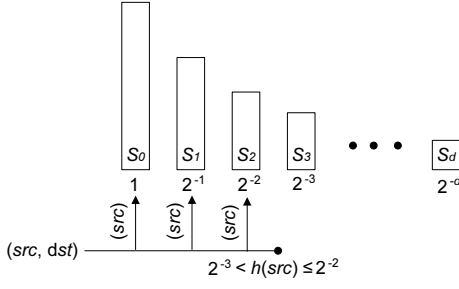
3

Figure 2: **Example recording process of the sampling algorithm**

Figure 3: **Example recording process of the coupon collecting algorithm**

Let F be a (ϵ', δ')-approximate distinct counting algorithm, *i.e.*, given a stream that contains X unique elements, the algorithm F outputs an estimate $\hat{X}$ for X, and $Pr[|\hat{X} - X| \leq \epsilon' X] > 1 - \delta'$. Let h be a uniform random hash function that maps (src, dst) pairs to $[0, 1)$. That is, each input is equally likely to map to any value in $[0, 1)$.

4.1.1 Sampling Algorithm

Figure 2 shows an example how the sampling algorithm works. Within each time interval, when a packet with a source-destination pair (src, dst) arrives, we first compute $x := h(src, dst)$. For $0 \leq i \leq d$, if $x < z^{-i}$, we insert the source src into the substream S_i. In this way, we sample each packet and create $d+1$ substreams $S_0, \ldots, S_d$. We then use the (ϵ', δ')-approximate distinct counting algorithm F to estimate the number of distinct sources in each substream S_i, denoted as l_i. Our algorithms are based on extracting information about the histogram vector m_i from the vector l_i. These techniques include either computing a complete approximation to the m_i vector, or detecting the presence of a substantial change in m_i.

Let L denote the vector of $(l_0, \ldots, l_d)$, and let M denote the vector of $(m_0, \ldots, m_d)$. We denote the expected value of l_i as $\bar{l}_i$, and denote the vector of $(\bar{l}_0, \ldots, \bar{l}_d)$ as $\bar{L}$.

Assume a source s connects to r unique destinations within the given time interval. Thus, s will be sampled into the substream S_i with probability $1 - (1 - \frac{1}{z^i})^r$ and variance

$$(1 - \frac{1}{z^i})^r (1 - (1 - \frac{1}{z^i})^r). \tag{1}$$

Therefore,

$$\bar{l}_i = \sum_r q_r (1 - (1 - \frac{1}{z^i})^r) \tag{2}$$

Based on the relationship between m_i and q_n, we have:

$$\sum_{0 \leq j \leq d} m_j (1 - (1 - \frac{1}{z^i})^{z^j}) \leq \bar{l}_i \leq \sum_{0 \leq j \leq d} m_j (1 - (1 - \frac{1}{z^i})^{z^{j+1}}) \tag{3}$$

From this analysis, we see that the l_i vector and the m_i vector are closely related. To compute approximate histograms, we can use linear programming to obtain feasible approximations for each m_i. However, in practice this technique yields a wide space of values for each bucket. In Section 4.2 we discuss heuristic methods for effectively obtaining information about the m_i distribution for the detection of anomalies. Further analysis on the effect of spreaders on
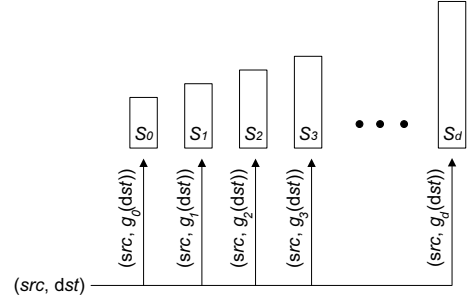
the sample values l_i are given in the Appendix A.

4.1.2 Coupon Collecting Algorithm

The variance of the random variables l_i in the first algorithm is high relative to expectation. This potentially leads to inaccurate estimations. The second sampling algorithm provides a modified technique designed to reduce the variance and provide better estimations.

When a packet with a source-destination pair (src, dst) arrives, we first use d uniform random hash functions (*i.e.*, $g_i(dst)$) each hashing dst to an integer in $[1, z^i], 0 \leq i \leq d$. We create d empty substreams $S_i, 0 \leq i \leq d$. For $0 \leq i \leq d$, we get $D_i = g_i(dst)$ and concatenate src and D_i into a new number $SrcD$, and then insert $SrcD$ into substream S_i. Figure 3 visualizes how the coupon collecting algorithm processes every input. We then use an (ϵ', δ')-approximate distinct counting algorithm F to estimate the number of distinct sources in each substream S_i, denoted as k_i. Assume the number of sources that communicate with r different destinations is q_r. We then have a relationship between k_i and q_r . Consider a source src which has r distinct destinations. For substream S_i and the r destinations, we insert at least 1 and at most $min(z^i, r)$ different $SrcD$ numbers into S_i. Let $X_{i,r}$ be the random variable that shows how many different numbers will be added into S_i. Counting $X_{i,r}$ is actually a variant of the classical coupon collecting problem [13]. One can imagine that a src has z^i kinds of coupons and one $g_i(dst)$ is a kind of collected coupon. Therefore, we have that [13]:

$$E(X_{i,r}) = z^i (1 - (1 - \frac{1}{z^i})^r) \tag{4}$$

$$VAR(X_{i,r}) = \begin{array}{l} (z^i - 2)(z^i - 1)(\frac{z^i - 2}{z^i})^{r-1} + \\ (z^i - 1)(\frac{z^i - 1}{z^i})^{r-1} - (z^i - 1)^2 (\frac{z^i - 1}{z^i})^{2r-2} \end{array} \tag{5}$$

Therefore, the expectation of k_i, $\bar{k}_i$ is:

$$\begin{aligned} \bar{k}_i &= \sum_r q_r E(X_{i,r}) \\ &= \sum_r q_r \cdot z^i (1 - (\frac{z^i - 1}{z^i})^r) \\ &= z^i \cdot \bar{l}_i \end{aligned} \tag{6}$$

From the above equation, we can see that the coupon collecting algorithm and sampling algorithm have the same expected output (if we ignore a constant factor). However, the variance of the outputs of the two algorithms are different (See Eq. 1 and Eq. 5). Numerically, the coupon collecting algorithm has much smaller variance than the sampling algorithm, making it a much more accurate estimator.

4

4.2 Spreader Detection Phase

In this section, we consider the two separate problems of reconstructing the outdegree histogram and of directly detecting stealthy spreaders. Our primary technique for histogram reconstruction is to use linear programming. We also discuss an alternate heuristic for reconstruction. For stealthy spreader detection, we apply a linear regression based heuristic to report information about any stealthy spreaders present in the data stream.

4.2.1 Outdegree Histogram Construction

To reconstruct the outdegree histogram, we rely on solving a linear program. Additionally in this section we discuss a possible alternate heuristic for histogram reconstruction.

Linear Programming Method.

From Eq 3, we have the linear inequality system:

$$B \times M \leq \bar{L} \leq A \times M, \text{where} \qquad (7)$$

$$A = [a_{ij}] = (1 - \frac{1}{z^i})^{z^j} - (1 - \frac{1}{z^i})^{z^{j+1}}, \qquad (8)$$

$$B = [b_{ij}] = (1 - \frac{1}{z^i})^{z^{j-1}} - (1 - \frac{1}{z^i})^{z^j}, \qquad (9)$$

Due to the sampling error and counting error from the (ϵ', δ')-approximate distinct counting algorithm F, the values of l_i deviate from the expected value $\bar{l}_i$ slightly, for $0 \leq i \leq d$. However, from l_i we can estimate the lower and upper bound of $\bar{l}_i$ [10]. Thus, we have an additional linear inequality system for $\bar{L}$:

$$\frac{l_i}{(1+\epsilon')(1+\epsilon_i)} \leq \bar{l}_i \leq \frac{l_i}{(1-\epsilon')(1-\epsilon_i)}, \text{for } 0 \leq i \leq d, \ (10)$$

where $\epsilon_i = \sqrt{\frac{3\ln(2/\delta')}{l_i}}$, for $0 \leq i \leq d$.

This linear inequality system from Eq. (7) and Eq. (10), together with the constraints $\bar{l}_i \geq 0$, $m_i \geq 0$, for $0 \leq i \leq d$, defines a convex hull. We can use linear programming to find a bound on this convex hull, by finding lower and upper bounds for m_i for $0 \leq i \leq d$. In particular, for each i we can solve the problem with the objective function set to minimize m_i, and denote the output lower bound for m_i as u_i, and the output upper bound for m_i as v_i. A natural heuristic is to compute the mean of the upper and lower bound derived for each m_i. Our experiments show that this heuristic approach obtains a reasonably accurate histogram estimation for normal network traffic, but it fails to accurately estimate the outdegree histogram for anomalous traffics (See Section 5.2).

Minimum Distance Method.

Another approach is to find a vector $\tilde{M}$ such that the distance between $A\tilde{M}$ and the input L is minimized. This effectively discards the error factor introduced by sampling. The intuition here is that by discarding this error factor, we restrict our set of possible M to those that, in expectation, generate a $\tilde{L}$ that is close to the input L.

However, the matrix A is an ill-conditioned matrix whose condition number, *i.e.*, the ratio of the largest to smallest
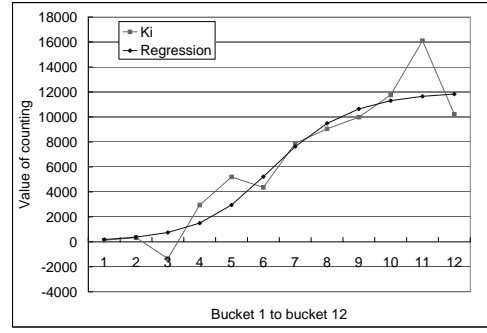

Figure 4: Example of linear regression.

singular value in the singular value decomposition, is very large [14]. Therefore, even when very small amounts of noise are introduced, the solution of the linear equations may differ a lot from the true values [15]. In other words, the estimation of M could be sensitive to noises and hence very inaccurate.

4.2.2 Stealthy Spreader Detection

As shown in Eq. 6, the expectation of k_i is determined by the outdegree distribution (*i.e.*, q_i) of the sources. We assume that in the change between two intervals of normal traffic, all q_i are 0 or very close to 0; while in the change between one interval of normal traffic and anomalous traffic, most q_i are close to 0 but have a peek around q_j. For example, a real scan event from a class B honeynet shows this feature (see Figure 7(a)). Therefore, we consider the ideal case that only one q_j is nonzero and propose this linear regression approach. However, it is worth mentioning that this approach works quite well for real cases.

Take Figure 4 as an example, which is computed from two continuous traffic intervals of CAIDA data (See Section 5.1 for details of the data). Using the algorithm described in 4.1.2 we obtain the values of $k_i (0 \leq i \leq d)$. Let MAX_D be the maximum outdegree occurring in a stealthy scan. For each $0 \leq j \leq MAX_D$, we use linear regression to find the q_j that fits to the values of k_i best. We use the mean squared error as the fitting metric, *i.e.*,

$$MSE_j = \sum_{0 \leq i \leq d} (k_i - q_j \cdot z^i (1 - (\frac{z^i - 1}{z^i})^j)^2. \qquad (11)$$

We take $\bar{j}$ such that $MSE_{\bar{j}} = \min_j MSE_j$ as our detection result and report that the spreaders have $\bar{j}$ outdegree on average.

We can also consider more complicated cases, *e.g.*, when there are two peeks in q_i's distribution. With a simple extension, we can compute the $\bar{j}_1$ and $\bar{j}_2$ that have the best linear regression in the sense of mean squared error. Generally, the best mean squared error becomes smaller as we allow more q_i to be nonzero. However, we are solving an ill-conditioned inverse problem, and the best mean squared error does not mean the best solution even when little noise is introduced. Once we allow many nonzero q_j values, the best linear regression result is very inaccurate. In practice, we believe most of the time there is one cooperative stealthy scan. Therefore we only try linear regression for one nonzero q_i and two nonzero q_i. If we do not see obvi-

ous improvement from using two nonzero q_i over using one nonzero q_i, we take the result of one nonzero q_i.

4.3 Memory Usage for Recording and Spreader Detection

Assuming the number of buckets and the source/destination space is a constant, our algorithms use constant memory. A (ϵ', δ')-approximate distinct counting algorithm typically uses $O(c \log(m))$ memory where c is a constant related to ϵ' and δ' and m is the maximum number of possible distinct elements (in our case 2^{64}, the total number of all possible source-destination pairs). In particular, 1KB memory is enough to achieve within 5% standard error using the probabilistic counting algorithm introduced in [10]. We use $2(d+1)$ ($d = 12$ in our experiments) data structures for distinct counting, and therefore we use constant memory which is $O(c \cdot d \cdot log(m))$ (*i.e.*, about 24KB in experiments).

In contrast, schemes such as [16] maintain for each source a distinct counting data structure. This yields a memory usage of $O(Sb)$, where b is the size of the distinct counting data structure and S is the number of distinct sources in the data stream. In the worst case, S can be as large as N, the size of the entire data stream.

A second technique for detection of stealthy spreaders is to use the superspreader technique [7] set at a threshold small enough to detect stealthy attacks. This technique is capable of reporting each key that contacts between $\frac{N}{2k}$ and $\frac{N}{k}$ with probability at least $1 - \delta$ using memory at most $O(\frac{N}{k} \ln \frac{1}{\delta})$. However, to detect stealthy attacks requires setting k to be small, making the memory usage closer to N. Thus, this technique is infeasible for the detection of stealthy spreaders.

5. EVALUATION

5.1 Evaluation Methodology

In this section, we evaluate our detection system with OC-48 backbone traffic traces collected by the Cooperative Association for Internet Data Analysis (CAIDA) on August 14th, 2002. The data contains packet headers collected at large peering points, and consists of about 3-hour 100G anonymized tcpdump records. The average packet rate is 191K/s. The average flow rate is 3.75K/s, and standard deviation is 0.16K/s.

We also consider a real scanning event collected from one class B honeynet. The scanning event happens on Jan 7th, 2006, and targets port 23. It lasts 2.5 hours, and has 1,607 unique sources, and 1,700,236 scan sessions. On average, each source sends 7 scans/min.

We have set the time interval to be 5 minutes and 1 minute in the experiments. Due to the difference of length between the two time intervals, the outdegree we are interested for stealthy spreader detection is different. For example, the outdegree we are interested in for the 5 minutes interval is from 2^4 to 2^{10}, while it changes to 2^3 to 2^9 for the 1 minute interval. Since there is no substantial change in the accuracy of the detection results, unless denoted otherwise, the default time interval for the following evaluations is 5 minutes. We apply two methods to simulate stealthy
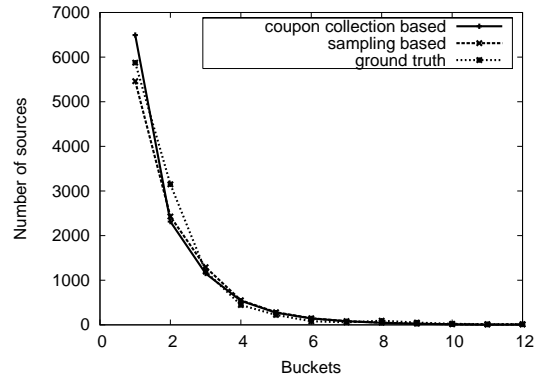

Figure 5: **Comparison of the real outdegree histogram with the estimated outdegree histogram by two algorithms**

spreaders in evaluation section. With the background normal traffic traces from CAIDA, the first method is to synthetically inject different intensities of scanning events into CAIDA traces. The second method is to insert a real scanning event collected from one class B honeynet into the CAIDA traces.

For both the sampling based and coupon collecting algorithms, we set $z = 2$, and $d = 12$. In our evaluation, we also try different settings for the parameter z and d, such as $z = \sqrt{2}$, $d = 24$, and $z = \sqrt[4]{2}$, $d = 48$. The detection accuracy is almost the same for these parameter changes. Thus we use $d = 12$ as the default setting to demonstrate the results below. The data recording phase of the system builds online the CDC and SDC structures for each time interval. At the end of each interval, we subtract SDC from CDC, and obtain a UDC structure. Stealthy spreader detection is performed to estimate the outdegree and intensity of spreaders. We then apply the method of *exponentially weighted moving average (EWMA)* to estimate the forecast error and determine whether there exists a stealthy spreader.

The evaluation metrics we used in this section include the histogram reconstruction accuracy, detection accuracy, and online performance. In particular, for detection accuracy, we consider relative error to measure the accuracy of outdegree estimation and the intensity estimation. The online performance metric consists of memory consumption, memory accesses per packet, and recording and detecting speed.

5.2 Histogram Reconstruction Evaluation

While it is not necessary to build the outdegree histogram from the network data stream to perform stealthy spreader detection, we believe accurate histogram approximation is still important and may have other applications. In this section, we compare the accuracy of reconstructed outdegree histograms for our two proposed algorithms with the ground truth.

Since traffic trends are similar for all intervals of normal traffic, we only show one interval example in Figure 5. From this figure, we observe that two estimated outdegree histograms are close to the real one, and estimation error of reconstruction for both algorithms is quite small for
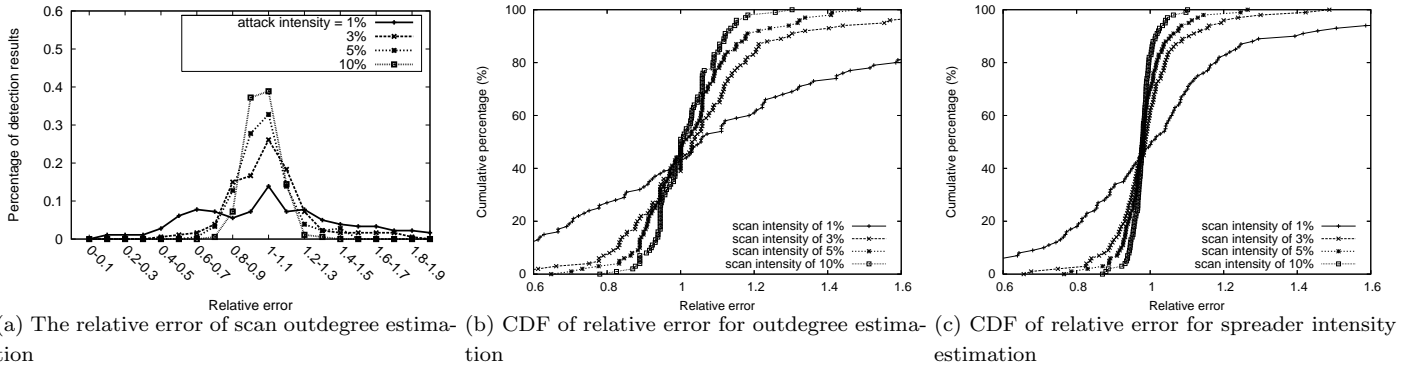
(a) The relative error of scan outdegree estimation (b) CDF of relative error for outdegree estimation (c) CDF of relative error for spreader intensity estimation

Figure 6: The effect of scan intensity ratio to detection accuracy for synthetic scans.

normal traffic. In order to show the accuracy of reconstruction, we further introduce Kullback-Leibler divergence $D_{kl}(P||Q) = \sum_i P(i)log\frac{P(i)}{Q(i)}$, which is a natural distance measure from a probability distribution P to a probability distribution Q. We calculate the divergence between the estimated histogram obtained from the sampling based algorithm and the real histogram obtained from accurate counting in all the intervals within 3-hour CAIDA data. As a result, the mean of divergence is 0.063, and the standard deviation is 0.023. Similarly, we calculate the divergence between the estimated histogram obtained from the coupon collecting algorithm and the real histogram. The mean of divergence is 0.049, and the standard deviation is 0.0028. This illustrates that the reconstruction works well for both algorithms within normal traffic, and the coupon collecting algorithm is more accurate than the sampling based algorithm. Moreover, the detection results based on interim structures also confirm that the coupon collecting algorithm is much better than the sampling based algorithm. Therefore, to save space, in the following evaluation we focus on the results for the coupon collecting algorithm.

While histogram reconstruction works well for normal traffic, it has large estimation errors when applied to anomalous traffic. Therefore, to improve speed and accuracy, we use the interim structures to detect spreaders, instead of estimating the values for the outdegree histogram in practice.

5.3 Evaluation of Spreader Detection

5.3.1 Threshold Setting of Change Detection

We apply the EWMA forecast model for change detection. By default, we use spreader intensity, which is the product of the number of scanners and their outdegree, as the forecast value for EWMA. We apply the same detection scheme to 33 5-minute intervals of normal CAIDA data traces. We remove three intervals with obvious changes and keep 30 intervals as a training dataset. According to the statistics of this training dataset, we set the threshold for intensity as 5000, which is the upper bound of intensity in this training dataset. On average, there are 1.1M flows in 5 minutes, and our threshold accounts for 0.45% of the total traffic flows in each interval. If the difference of the spreader intensity between the current interval and the previous interval is larger than our threshold, we will further use the outdegree estimation value. We mainly focus on anomalies,

which have outdegree larger than 2^4, and ignore the smaller spreaders, which may not be anomalies.

5.3.2 Simulation with Synthetic Stealthy Scan

In this section we show the relationship between detection accuracy and the injected scan intensity ratio. We define the scan intensity ratio as the ratio of total scan flows to the total flows of normal traffic in that interval. In order to show the detection accuracy affected by scan intensity ratio, we synthetically inject scanning flows into CAIDA traffic with scan intensity ratios from 1% to 10%, and change the spreader degree from bucket 5 (the lower bound 2^4) to bucket 10 (the upper bound 2^{10}). We show the effect of scan intensity ratio in Figure 6(a). Here the relative error is defined as the ratio of the estimated outdegree to the real spreader outdegree. From the figure, we can see that the accuracy for a 3% scan intensity ratio is much better than for a 1% scan intensity ratio. Since we choose the threshold of intensity as 5000, there is no false positive in our detection. The false negative rate for a 1% scan intensity is 17.8%, while that for a 2% scan intensity is 0. Combined with the concrete statistics in Table 1, with the scan intensity ratio increasing from 1% to 3%, probability of outdegree estimation error within 20% increases from 33.9% to 76.1%. In fact, when the scan intensity ratio is around 2%, we already achieve the satisfied accuracy.

Figure 6(b) and (c) show our detection results using linear regression. In Figure 6(b), the x axis is the relative error for outdegree estimation, and the y axis is the cumulative distribution of the relative error. With the scan intensity ratio increasing from 1% to 3%, the percentage of cases with relative error range of [0.8, 1.2] increases from about 30% to 90%. In Figure 6(c), the relative error ratio is defined as the ratio of the estimated spreader intensity to the real spreader intensity. With the scan intensity ratio increasing from 1% to 3%, the percentage of cases with relative error range [0.9, 1.1] increases from about 35% to 80%. These three figures show that our spreader detection approach can accurately detect the outdegree and the spreader intensity, when the spreader intensity ratio is larger than 2%.

### 5.3.3 Simulation with Real Stealthy Scan

We also evaluate our detection scheme with real stealthy scans from honeynet data. We insert the first 5-min data of a real scanning event collected from one class B honeynet into the normal CAIDA traces. Figure 7(a) shows the distri-

7

| Scan intensity	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
Percentage within 10% error	21.1%	30.6%	42.8%	51.7%	60.6%	67.2%	70.6%	72.2%	76.1%	76.1%
Percentage within 20% error	33.9%	58.9%	76.1%	83.9%	87.2%	90%	93.9%	95%	96.7%	97.8%
Percentage within 30% error	48.9%	78.9%	87.2%	91.7%	94.4%	96.1%	98.9%	98.9%	99.4%	99.4%
Percentage within 40% error	61.7%	85%	91.1%	95.6%	97.2%	99.4%	100%	100%	100%	100%
Percentage within 50% error	71.1%	89.4%	93.9%	99.4%	100%	100%	100%	100%	100%	100%

Table 1: Effect of scan intensity ratio.
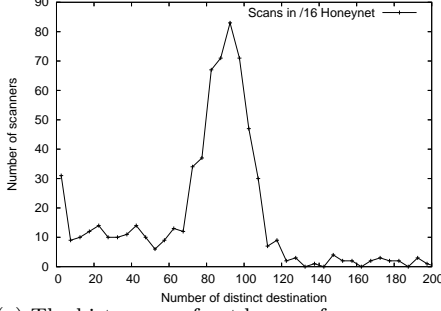
(a) The histogram of outdegree of scanners collected in the honeynet
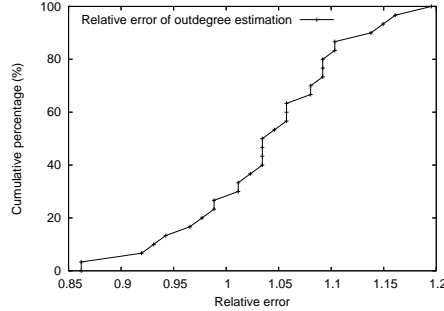
(b) CDF of relative errors of scan outdegree estimation
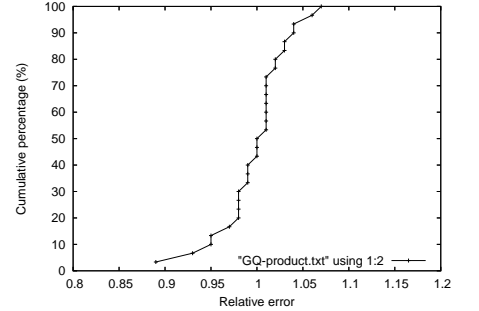
(c) CDF of relative error of spreader intensity estimation

Figure 7: The effect of scan intensity ratio to detecting accuracy for real scan.

bution of the outdegree (*i.e.*number of distinct destinations) of the scanners observed in the honeynet. It is clear that most scanners have a similar outdegree, which is around 90. In fact, the median outdegree of all the scanners is 87. Therefore, we define this event to be a stealthy spreader with outdegree of 87.

Figure 7(b) and (c) shows our detection results using linear regression mentioned in 4.2.2. Figure 7(b) shows that in about 80% of the cases, the relative error of outdegree estimation is within the range [0.9, 1.1], and even in the worst case the relative error is less than 20%. Figure 7(c) shows that in about 90% of the cases, the relative error of spreader intensity estimation is within the range [0.95, 1.05], and even in the worst case the relative error is less than 11%. Therefore, we believe our spreader detection approach can accurately detect both outdegree and spreader intensity for the stealthy spreader.

5.4 Online Performance Evaluation

5.4.1 Memory Consumption

It is very important to have small memory consumption for online traffic recording over high-speed links in order to make use of the fast SRAM, as well as to allow for potential implementation in specialized hardware, *e.g.*, FPGA or ASIC. In normal CAIDA traffic, on average there are 1M different source IPs within a five minute interval. By using our detection method, we only need a total memory of about 24KB for traffic recording. We use the algorithm introduced in [10] to do distinct counting. To achieve the (ϵ', δ')-approximate distinct counting algorithm where $\epsilon' = 0.05$ and $\delta' = 0.1$, each distinct counting data structure needs $1KB$ memory. In total we keep 24 distinct counting data structures for both CDC and SDC in our detection algorithm. The detection phase does not use more memory, and therefore we use about 24KB in total. On the other hand, if we use a more naive method and keep distinct counters for each source IP, we need $2 \times 1M \times 1KB = 2GB$ in

the extreme case. Thus, with the increase of unique source IPs, the naive method will run out of memory very quickly in this scenario.

5.4.2 Memory Access per Packet and Speed Results

Since we use the distinct counting algorithm introduced in [10], we have 1 memory access per packet. Although we have 24 distinct counting data structures, when establishing these histograms in parallel or in pipeline, our system is capable of online monitoring and stealthy scan detection.

In our experiments, we use a 3.2GHz Pentium 4 computer for recording and stealthy spreader detection. For each 5-min CAIDA data interval (about 1.1M packets), we take about 200 seconds to parse the data and maintain the data structures, and then take less than 0.1 seconds to detect whether there is stealthy spreader. Based on our speed results, we believe that the processing time can be substantially reduced with the addition of some specialized hardware (*e.g.* for fast hashing) and the application of parallelization techniques.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we propose the stealthy spreader detection problem and design an online outdegree histogram based detection system. The system has two phases: the recording phase and the spreader detection phase. In the recording phase, we propose two randomized algorithms which leverage distinct counting algorithms to estimate some aggregate metrics of the outdegree histogram. Then, in the detection phase, we propose the use of linear programming to reconstruct the outdegree histogram. Based on large, real network traces we show that the reconstructed outdegree histogram is quite accurate compared to the ground truth. We further propose a linear regression based approach for stealthy spreader detection, which does not directly rely on the histogram reconstruction. Simulated with real network traffic as the background traffic, our approach

can accurately find out both synthetic stealthy scans and real scans extrapolated from honeynet data. Further, our system requires a small, fixed size memory usage and has fast processing speed, which is quite desirable for an online monitoring and anomaly detection system.

Our work in this paper is primarily focused on detection of anomalous traffic patterns. However, considering online performance, e.g. memory usage, we do not keep per flow information, thus there is no source information retained for suspicious anomalies. Extending this work to consider mitigation is an important direction for future work. Given that our detection scheme can provide detailed information about the outdegree histogram of a suspected spreader, in particular the scan rate of a cooperative scan, it is plausible this information can be combined with offline correlation analysis and further sampling to mitigate the potential attack activities.

7. REFERENCES

[1] G. Cormode and S. Muthukrishnan, "What's new: Finding significant differences in network data streams," in *Proc. of IEEE Infocom*, 2004.
[2] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proc. of ACM SIGCOMM*, 2002.
[3] A. Chakrabarti, K. D. Ba, and S. Muthukrishnan, "Estimating entropy and entropy norm on data streams," in *Proceedings of the 23rd International Symposium on Theoretical Aspects of Computer Science (STACS)*, 2006.
[4] S. Guha, A. McGregor, and S. Venkatasubramanian, "Streaming and sublinear approximation of entropy and information distances," in *Proceedings of ACM Symposium on Discrete Algorithms (SODA)*, 2006.
[5] P. B. Gibbons, Y. Matias, and V. Poosala, "Fast incremental maintenance of approximate histograms," in *Proceedings of the 23rd VLDB Conference*, 1997.
[6] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss, "Fast, small-space algorithms for approximate histogram maintenance," in *Proceedings of STOC*, 2002.
[7] S. Venkataraman, D. Song, P. Gibbons, and A. Blum, "New streaming algorithms for fast detection of superspreaders," in *Proceedings of Network and Distributed System Security Symposium (NDSS)*, 2005.
[8] Q. Zhao, A. Kumar, and J. Xu, "Joint data streaming and sampling techniques for detection of super sources and destinations," in *Proceedings of Internet Measurement Conference (IMC)*, 2005.
[9] A. Lall, V. Sekar, M. Ogihara, J. Xu, and H. Zhang, "Data streaming algorithms for estimating entropy of network traffic," in *Proceedings of ACM SIGMETRICS*, 2006.
[10] P. Flajolet and G. N. Martin, "Probabilistic counting algorithms for data base applications," *J. Computer and System Sciences*, vol. 31, pp. 182–209, 1985.
[11] N. Alon, Y. Matias, and M. Szegedy, "The space complexity of approximating the frequency moments," *J. of Computer and System Sciences*, vol. 58, pp. 137–147, 1999.
[12] Z. Bar-Yossef, R. Kumar, D. Sivakumar, and L. Trevisan, "Counting distinct elements in a data stream," in *RANDOM 2002*, 2002.
[13] W. Feller, *An Introduction to Probability Theory and Its Applications*, 3rd ed. New York: Wiley, 1968.
[14] P. Hansen, *Rank-deficient and discrete ill-posed problems*. Philadelphia: the Society for Industrial and Applied Math, 1998.
[15] A. Tarantola, *Inverse Problem Theory*. the Society for Industrial and Applied Math, 2005.
[16] C. Estan, G. Varghese, and M. Fisk, "Bitmap algorithms for counting active flows on high speed links," in *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM Press, 2003, pp. 153–166.

APPENDIX

A. ATTACK EFFECT ON SAMPLING DATA STRUCTURE

To analyze when a change can be detected using the approximate histogram, we examine two quantities: (1) an upper bound on the change in the value of l_i when the number of items in each bucket does not change by more than c_l, and (2) a lower bound on the change in the value l_i when the number of items in the buckets changes by at least c_h. Together, these give bounds on how large c_h needs to be before an attack is guaranteed to be detected with high probability.x

Let $X_{i,s}$ be a random variable that denotes when source s gets added to bucket i: if source s gets added to bucket i, $X_{i,s} = 1$, otherwise, $X_{i,s} = 0$. Then $l_i = \sum_s X_{i,s}$, so $E[l_i] = \sum_s E[X_{i,s}]$.

In this analysis, we first get a bound on $E[l_i]$, and then we use Chernoff bounds to compute the ϵ such that $Pr[l_i > (1+\epsilon)E[l_i]] \leq \delta$, where δ is the acceptable probability of error that is given as input.

Let source s contact z^j destinations. (For simplicity, in the analysis, we will assume that all sources contact z^k destinations, for some integral value k). Note that

$$Pr[X_{i,s} = 1] = 1 - (1 - \frac{1}{z^i})^{z^j}.$$

Therefore, $E[l_i] \leq \sum_j (m_j(1+c_l))(1 - (1 - \frac{1}{z^i})^{z^j})$.

We can now use the Chernoff bound to calculate an upper bound on the change in l_i, using

$$Pr[l_i > (1+\epsilon)E[l_i]] \leq e^{\frac{-E[l_i]\epsilon^2}{4}}, \text{ for } \epsilon < 2e - 1.$$

Note that $\epsilon E[l_i]$ is the deviation in l_i above its expectation, so we can solve for it to get the following:

$$\epsilon E[l_i] \leq \sqrt{4 \ln \frac{1}{\delta} E[l_i]}. \tag{12}$$

We can do a similar analysis to get a bound on l_i in the presence of an attack, using the information of bucket i. Now we use the Chernoff bound to calculate a deviation below in l_i from its expectation. The bound states that

$$P[l_i < (1-\epsilon)E[l_i]] \leq e^{\frac{-\epsilon^2 E[l_i]}{2}}.$$

Under the attack scenario, the $E[l_i]$ changes, increasing from the previous value in the following manner:

$$E[l_i] \geq \sum_j (m_j + y_j c_h)(1 - (1 - \frac{1}{z^i})^{z^j}),$$

where $y_j = 1$ if bucket j contains an attack.

Using the Chernoff bound again, we can compute the minimum value that l_i will have with probability at least $1 - \delta$.

$$\epsilon E[l_i] \leq \sqrt{2 \ln \frac{1}{\delta} E[l_i]},$$

Therefore, $(1-\epsilon)E[l_i] \geq E[l_i](1 - \sqrt{2 \ln \frac{1}{\delta}})$.