PAPER

Detecting Stealthy Spreaders by Random Aging Streaming Filters

MyungKeun YOON[†a)], *Member and* Shigang CHEN[††b)], *Nonmember*

SUMMARY Detecting spreaders, or scan sources, helps intrusion detection systems (IDS) identify potential attackers. The existing work can only detect aggressive spreaders that scan a large number of distinct destinations 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 current IDS's have serious limitations. Being lightweight, the proposed scheme can detect scan sources in high speed networking while residing in SRAM. By theoretical analysis and experiments on real Internet traffic traces, we demonstrate that the proposed scheme detects stealthy spreaders successfully.

key words: *network security, intrusion detection, spreader detection, port scan, anomaly detection*

1. Introduction

Cyber attacks are commonly preceded by a reconnaissance phase that probes the target network with address scans for live hosts, port scans for live ports, and platform scans for software/hardware information, which leads to the identification of vulnerability to be exploited. Guarding against cyber attacks, the first line of defense is to detect those scan sources. It provides the list of potential offenders.

The operations of intrusion detection are expensive. The problem becomes much more serious when real-time traffic analysis is required for an intrusion detection system (IDS) at the high-speed gateway of a large enterprise network. In this case, it is important for the system to perform differentiated analysis, focusing more resources on traffic from sources that are more likely to be offenders. Scan detection is one of the tools that provides the IDS with the list of potential offenders before the actual attacks occur. We will use "intrusion detection," "scan detection," and "spreader detection" interchangeably.

As intrusion detection systems are deployed, attackers also invented evasive techniques. To evade scan detection, an attacker may send packets with a long interval between them. As the resource of detection system is limited, the history of attack packets would be flushed periodically and hence no alarm would be generated. The waiting time for the attack would increase, which would bother attackers. However, recent cyber attacks pursue financial profit; the attackers with motivation would like to endure such inconvenience. In this paper, we tackle this problem and propose a new defensive scheme to detect such an evasive scanning by crafty attackers.

Consider the packet flow from the Internet side of a gateway to the intranet side. Each packet represents a *contact* (a, b), where a is a source on the Internet and b is a destination on the intranet. The *source* (*destination*) may be defined as an IP address, a port number, or an address/port pair. We define *cardinality* of a source to be the number of *distinct* destinations that the source contacts. Similarly, we define destination cardinality to be the number of *distinct* sources that contact the destination. A group of sources is called a *top-contact set* if the cardinality of any source outside of the group is smaller than the cardinality of any source inside the group. The problem of *spreader detection* is to identify a top-contact set. In this sense, detecting scan sources is under the category of spreader detection problem [1], [2]. The research community and industry have proposed a variety of schemes for spreader detection, which are summarized in Sect. 5.

We notice that current IDS's have several limitations in identifying spreaders. First, a complete data loss occurs whenever a detection period ends and a new one begins. We notice that current IDS's use time-windows to detect spreaders. They divide the whole monitoring time into smaller non-overlapping time-windows (monitoring periods)[*]. IDS's approximately estimate the cardinality of any source based on the information collected during the previous time-window. A time-window may be defined as either a fixed length of time such as 10 seconds or 5 minutes, or a variable length such that a time-window finishes when the memory becomes full. In either case, when the current time-window finishes, all stored information is erased from the memory and a new time-window begins with a clean state. This approach surely causes detection errors due to the information gap between the consecutive time-windows. Suppose that an IDS is supposed to trigger a detection alarm when the estimated cardinality of any source is above a predefined threshold. If a spreader contacted half of the threshold destinations in one monitoring period and the other half in the next monitoring period, it would not be detected. In this paper, we call such a spreader a *stealthy*

Manuscript received May 24, 2010.
Manuscript revised January 13, 2011.

[†]The author is with the Department of Computer Engineering, Kookmin University, Seoul, Korea.
[††]The author is with the Department of Computer & Information Science & Engineering, University of Florida, Gainesville, USA.
a) E-mail: mkyoon@kookmin.ac.kr
b) E-mail: sgchen@cise.ufl.edu

DOI: 10.1587/transcom.E94.B.2274

[*]"time-window" and "monitoring period" will be used interchangeably.

spreader. Stealthy spreaders can happen accidently due to the incompleteness of the time-window based detection, or smart attackers can cause this problem on purpose as they send scan packets slowly and steadily.

The second limitation is that the memory space of IDS is quickly filled up with meaningless normal traffic rather than scan packets. This is because the volume of normal traffic overwhelms that of any spreader. If we can store this less significant information about normal traffic within a small portion of the memory, we can use the other spaces to record more important information about the scan packets. Note that the spreader detection should be implemented in SRAM to catch up with the speed of a high-speed network. As of now, SRAM is an expensive resource.

In this paper, we propose a new spreader detection scheme to overcome all the limitations described above. The proposed scheme can detect stealthy spreaders and minimize the negative effects of normal packets. It is lightweight, so it can detect scan sources in high speed networking while residing in SRAM.

The rest of the paper is organized as follows. Section 2 presents the design of our detection schemes. Sections 3 and 4 evaluate the proposed scheme via theoretical analysis and experiments with real Internet traffic traces, respectively. Section 5 discusses the related work. Section 6 draws the conclusion.

2. Stealthy Spreader Detection

The basic unit of our system is a two-dimensional bit array, which we call a filter or *random aging streaming filter* (RAS). We first explain *RAS* in detail. Then, we propose a heuristic method to configure parameters for *RAS*.

2.1 Random Aging Streaming Filter (RAS)

We assume that the cardinality of any spreader is above a predefined threshold θ and the cardinality of any normal source is below θ. The goal of *RAS* is to find any spreader.

The random aging streaming filter (RAS) uses an $M = n \times m$ bit array as its main data structure, which is initialized to be all zeros. We first give some definitions. Each bit $B(x, y)$ is referenced based on a row index x and a column index y. 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 table that are set as one. Similarly, the fullness ratio of row x is defined as $\frac{c(x)}{m}$. We define a *system parameter* α, specifying the desirable fullness. If $R > \alpha$, the *aging procedure* will kick in to randomly reset some bits to zeros in order to bring R down. Figure 1 illustrates an example of RAS.

Next we describe the operations of *RAS*. When receiving an input contact (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

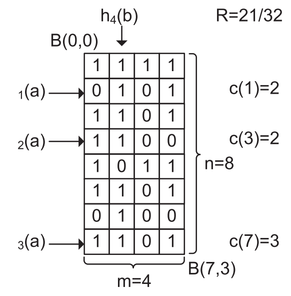


Fig. 1 An example of random aging streaming filter ($k = 3$).

is $[0...m-1]$. The filter sets k bits, $B(x_1, y)$, ..., $B(x_k, y)$, to be one. Note that the row indexing becomes a Bloom filter [3], [4]. 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, we further check whether a is a spreader or not as follows. We will show how β should be determined in the following section.

1. Let I_j be an indicator random variable for column j. We set I_j to be one if $B(x_i, j) = 1$ for all the representative rows of a; otherwise, $I_j = 0$. We define a_r to be $a_r = \sum_{j=0}^{m-1} I_j$.
2. Estimate the cardinality of a, denoted as $\hat{a_c}$, by using equation 1.
$$\hat{a_c} = m \times \ln\left(\frac{m}{m - a_r}\right) \tag{1}$$
3. If $\hat{a_c}$ is above θ, we consider source a a spreader.

Equation (1) is from [5], but we check the membership of a against all the columns, as in step 1. Note that computing I_j equals the Bloom filter membership test [3]. As some I_j's may have been set to ones due to Bloom filter errors, $\hat{a_c}$ may not be less than the real cardinality. There are advantages and disadvantages in using $\hat{a_c}$ as the estimator. The first advantage is that we can detect spreaders aggressively so that the number of undetected spreaders can be reduced. Second, the estimation process can be implemented efficiently, which will be explained later. However, the disadvantage is that more false positives will be triggered. To suppress these false alarms, we set β to be large enough, which will be explained in the next section.

Note that we define the column index to be $y = h_{k+1}(b)$, which is different from [1] that uses $y = h_{k+1}(< a, b >)$. This choice helps RAS to minimize the negative effects of normal traffic. Suppose b represents a busy webserver in an enterprise network and billions of clients connect b. If $y = h_{k+1}(< a, b >)$ is used, these clients will fill up the whole bit table since a randomizes y. To the contrary, only one column of the bit table will be filled up with one-bits if $y = h_{k+1}(b)$ is used. We want to point out that our choice will cause more collisions within the bit table. These collisions may cause more false positives. While designing RAS, we trade the estimation accuracy for cardinality with the better detection of spreaders. In the next section, we will show that this estimation inaccuracy can be controlled by setting α appropriately.

After some bits are set as one, if R becomes greater than α, the filter performs the aging procedure as follows. It randomly selects a column and zeros all bits in the column. It repeats the above operations until $R \leq \alpha$. We call it *random aging*. Note that column-based deletion does not break consistency of any packet's information in the bit table since the packets of same contact should be programmed into the same column.

We note that RAS detects stealthy spreaders as well as normal spreaders. This is because only a small portion of the bit table is cleared out when it becomes full. The temporary bursty normal traffic cannot delete much information about the scan sources. Additionally, the random aging just resets one column at a time, which decreases $c(x)$ by one at most for each row x in the bit table. Unless the bursty traffic lasts for a long time, RAS can persistently keep the information about scan sources.

2.2 Parameter Configuration

Although θ and M are already determined, the performance of *RAS* still depends on several parameters such as α, β, m, n, the number of spreaders, and even traffic patterns. We propose a heuristic method to configure these parameters.

We first assume the ideal case where no collision occurs among the rows in the bit table. We say that a row collision happens when two different sources map into any same representative row in the bit table. This assumption is not true since collisions may frequently happen in practice. However, we note that row collisions may increase only false positives not false negatives in the bit table. Therefore, we first determine such parameters as m and β so that false negatives hardly happen under this assumption[†]. Once these parameters are determined, we consider the next case where collisions occur among the rows in the bit table. Under this case, we determine α so that false positives hardly happen. Therefore, we can reduce both false positives and false negatives by appropriately setting m, β, and α, sequentially.

2.2.1 Assuming No Row Collisions

We first consider an ideal case. We assume that no row collisions happen in the bit table. Then, we can detect spreader a if all the representative rows satisfy Eq. (2). Note that θ is the given threshold value for spreaders.

$$\theta < m \times \ln\left(\frac{1}{1 - \frac{a_r}{m}}\right). \tag{2}$$

We define β to be $\frac{a_r}{m}$ when $\theta = m \times \ln(\frac{1}{1-\frac{a_r}{m}})$ holds. The meaning of β is the ratio of one-bits in a row when the estimated cardinality is equal to θ. By definition, β becomes

Table 1 Parameter configuration examples (c = 9).

θ	100	200	300	400	500	600	700	800
β	0.79	0.79	0.90	0.79	0.86	0.90	0.94	0.79
m	64	128	128	256	256	256	256	512
α	0.28	0.40	0.51	0.51	0.58	0.63	0.67	0.60

$$\beta = 1 - e^{-\frac{\theta}{m}}. \tag{3}$$

We use Eq. (3) to determine m and β. A problem is that neither m nor β is determined yet, so the equation can have an infinite number of solutions. However, the equation gives a hint on how we should determine β. Note that small m is desirable for RAS. This is because small m allows large n, which reduces the row collisions in the bit table. However, if β is very close to one, there is a high probability that a random aging fails the spreader detection. We explain it as follows. Suppose $\beta \approx 1$ and a random aging happens. Then, we need another $m \times \log(m)$ connections from the same source to increase $c(x)$ of the representative rows. Note that $m \times \log(m)$ is from the coupon collection problem [6]. Through experiments, we confirmed that few spreaders could be detected when β was close to one. In this paper, we choose β to be below 0.95, but it can be adjusted according to the specific application or environment. Once β is chosen, m can be determined according to Eq. (3).

Alternatively, we may set m first and determine β accordingly. For example, it would be a good choice to have m as a multiple of two, which helps to optimize memory allocation. In this paper, we choose m to be a multiple of a word, i.e. 32 bits, 64 bits, 128 bits, and so on. For each m, we compute β and choose the largest β below 0.95. Choosing the largest β minimizes m. Table 1 shows some examples for parameter configuration. It shows how β and m are determined for θ from 100 to 800. We explain how to determine α in the next subsection.

2.2.2 Assuming Row Collisions

We assumed that no row collisions occurred when determining β and m. With these parameter values, spreaders can be caught without triggering false negatives. However, normal sources may be mistaken as spreaders when row collisions happen. This is because we set β to be rather loose in order to detect spreaders aggressively. Therefore, we need to set α appropriately so that false positives should be reasonably suppressed. We can guess how α affects the detection by intuition. If α is too small, false positives may hardly happen, but random aging may occur aggressively. Then, information loss causes false negatives. To the contrary, if α is too large, RAS may trigger lots of false positives. We will use some statistical properties to determine α.

Suppose that only normal sources have been witnessed for a quite long period of time. Then, the bit table has been

[†]Once m is determined, n is also fixed because $M = n \times m$.

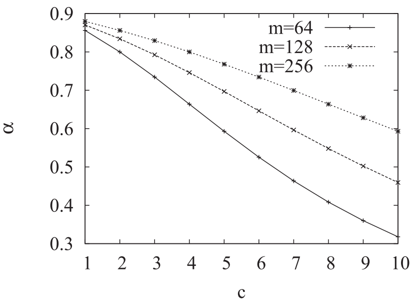


Fig. 2 Curve of α with respective to c ($\beta = 0.9$).

populated with the normal traffic. Let Y be a random variable to represent $c(x)$ for row x in the bit table. Then, the following equations hold for the expectation and the variance of Y, which we prove in the appendix Appendix.

$$E(Y) = \alpha \times m \tag{4}$$

$$V(Y) = \alpha \times m \times (1 - \alpha) \tag{5}$$

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

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

where statistical errors become small with large c. Equation (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. The probability depends on the value of c. To the contrary, if x is a representative row for any spreader, $c(x)$ should be larger than $U(Y)$. By setting $U(Y)$ to be $\beta \times m$, we have the following equation.

$$\beta \times m \ = \ \alpha \times m + c\sqrt{\alpha \times m \times (1 - \alpha)} \tag{7}$$

Solving for α, we have

$$\alpha \ = \ \frac{2\beta m + c^2}{2(m + c^2)} - \sqrt{\left(\frac{2\beta m + c^2}{2(m + c^2)}\right)^2 - \frac{m\beta^2}{m + c^2}}. \tag{8}$$

Figure 2 shows how α changes with c when β is 0.9. Table 1 shows α as a result of the proposed heuristic method to configure β, m, and α.

3. Analysis of the Random Aging

Random aging deletes only a small portion of the bit table, which improves the detection of stealthy spreaders. In this section, we demonstrate it formally through a probability model. We show that RAS with the random aging performs better than RAS with no aging, which are denoted by RAS_A and RAS_N, respectively[†]. Note that the whole bit table is reset in RAS_N while only one column is reset in RAS_A. We compare the numbers of 1-bits of RAS_A and RAS_N when an arbitrary contact is given. The idea is that more information about stealthy spreaders would be available if there are more 1-bits in the bit table.

In this probability model, we assume that the source and destination of a contact is randomly chosen for simplicity. We also assume that only one hash function is used for the row index (i.e. $k = 1$) while $k = 3$ is used for the real experiments. Although this analytical model shows why the random aging improves the detection, it has limitations of not reflecting real traffic distribution and not using $k = 3$. To supplement this, we also did experiments and show the results in the next section.

Suppose that enough time has passed since the detection algorithm started. Now, let's consider a specific packet that just arrives and is about to be programmed into the bit table, which is denoted by $pkt = (a, b)$. We define a random variable U to denote the number of 1-bits in the bit table when pkt arrives. We derive the probability $P(U = i)$ and compute $E[U]$. We first solve it for RAS_N and then for RAS_A. The comparison confirms that more 1-bits are available when RAS_A is used.

3.1 RAS_N

For RAS_N, U has the range of $[0...\alpha \times n \times m]$. To derive $P(U = i)$, we define another random variable S_i, which denotes the average number of packets required to change U from i to $i + 1$. Since the probability to set a bit from zero to one is $1 - \frac{i}{n \times m}$, we have

$$S_i \ = \ \frac{1}{1 - \frac{i}{n \times m}}$$

Assuming that packets arrive at a constant rate, we can use the accumulated value of S_i as the elapsed time. Figure 3(a) shows how U changes as the accumulated number of packets increases in RAS_N. From this, we can derive $P(U = i)$ as the ratio of S_i to the total sum of S_i. Therefore,

$$P(U = i) \ = \ \frac{S_i}{\sum_{j=0}^{\alpha \times n \times m} S_j} \approx \frac{S_i}{nm \ln\left(\frac{1}{1-\alpha}\right)}$$

where $\sum_{j=t}^{\alpha \times n \times m} S_j \approx \int_t^{\alpha \times n \times m} \frac{1}{1 - \frac{x}{nm}} dx = n \times m \times \ln(\frac{1 - \frac{t}{nm}}{1-\alpha})$ [7]. We can derive $E[U]$ as follows:

$$\begin{aligned} E[U] \ &= \ \sum_{i=0}^{\alpha \times n \times m} i \times P(U = i) \\ &= \ \sum_{i=0}^{\alpha \times n \times m} i \times \frac{1}{(nm - i) \times \ln\left(\frac{1}{1-\alpha}\right)} \end{aligned} \tag{9}$$

3.2 RAS_A

Since U of RAS_A has the range of $[\alpha \times n \times (m-1)...\alpha \times n \times m]$, $P(U = i)$ and $E[U]$ become as follows:

$$P(U = i) = \frac{S_i}{\sum_{j=\alpha \times n \times (m-1)}^{\alpha \times n \times m} S_j} = \frac{S_i}{nm \ln\left(\frac{1 - \frac{\alpha(m-1)}{m}}{1-\alpha}\right)}$$

$$E[U] = \sum_{i=\alpha \times n \times (m-1)}^{\alpha \times n \times m} i \times \frac{1}{(nm - i) \times \ln\left(\frac{1 - \frac{\alpha(m-1)}{m}}{1-\alpha}\right)} \tag{10}$$

[†]Throughout this paper, RAS means RAS_A except this section.

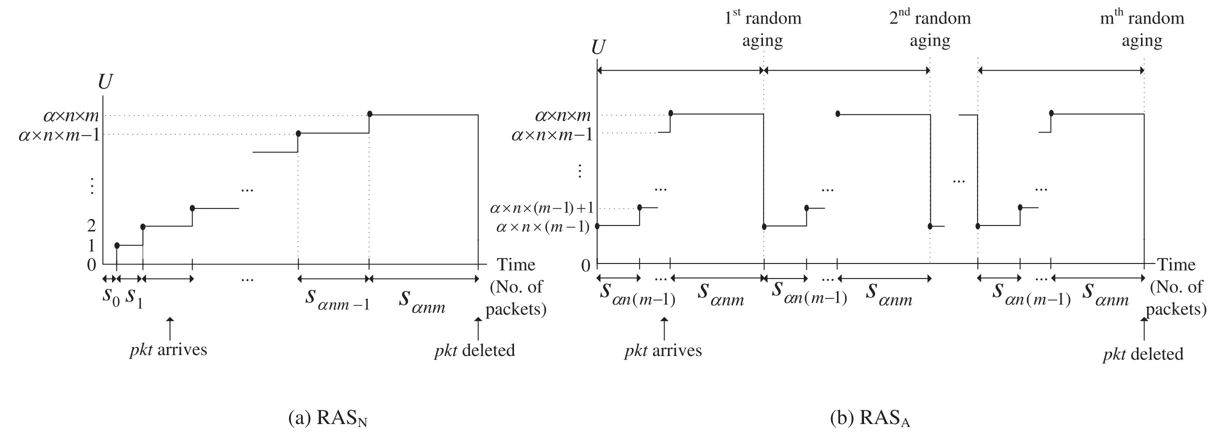


Fig. 3 Number of 1-bits (U) in the bit table when *pkt* arrives. We use the accumulated number of packets as the elapsed time for the x axis.

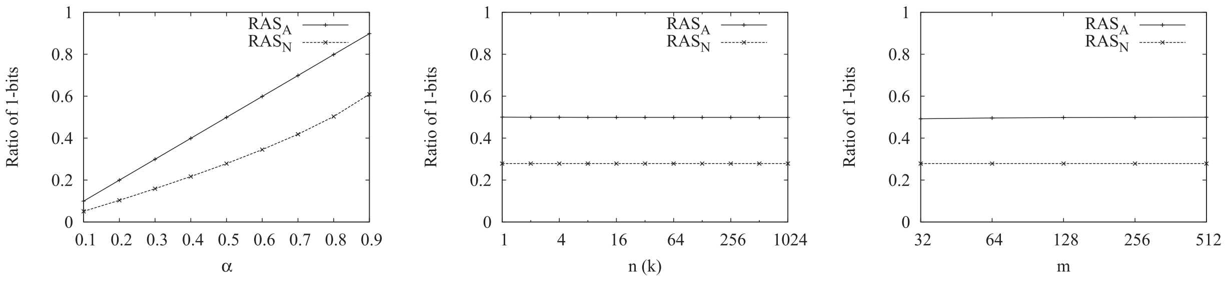


Fig. 4 The ratio of 1-bits in the bit table when *pkt* arrives. The ratio is equal to $\frac{E[U]}{nm}$. A large ratio is helpful to detect stealthy spreaders. The default parameters are $\alpha = 0.5$, $n = 16,384$ and $m = 256$.

where $\alpha \times n \times (m-1) \leq i \leq \alpha \times n \times m$.

Figure 3(b) shows how U changes as the accumulated number of packets increases in RAS_A. Note that U ranges from $\alpha \times n \times (m-1)$ to $\alpha \times n \times m$.

Figure 4 shows how the ratio of 1-bits changes, which equals $\frac{E[U]}{nm}$. The value of α, n, and m changes for each graph. We use $\alpha = 0.5$, $n = 16K$, and $m = 256$ as default parameters. Note that the large ratio of 1-bits is desirable to detect stealthy spreaders. In the first graph, α changes from 0.1 to 0.9. We see that RAS_A increases the ratio by 47% ∼ 97%. In the second and third graphs, n and m change respectively. In both cases, the random aging increases the ratio by around 80%.

4. Experiment

We evaluate RAS through experiments using real Internet traffic traces. We implement RAS and the online streaming module (OSM) [1] for comparison as these two streaming filters are lightweight and can reside in SRAM. The experimental results show that RAS detects spreaders better than OSM while triggering a smaller number of false positives.

Like RAS, the data structure of OSM is based on a two-dimensional bit array. The row index is computed the same way with RAS, but the column index is further randomized by using both the source and destination addresses. This means that the bit array can be completely filled up with all packets from only a single source. We emphasize that OSM does not have any aging scheme; once the bit array is populated, all the contents will be lost with a reset. Actually, RAS tackles these limitations of OSM. For more details, readers should be referred to [1].

We use packet traces collected from the gateway router at a campus network for 24 hours. We take only the inbound session from the Internet. The trace contains 751,286 distinct source IP addresses, 120,916 distinct destination IP addresses, 445,926,847 packets and 2,427,327 distinct contacts. Note that a contact is denoted by (a, b) where a/b is the source/destination IP address of a packet. In this sense, the goal of the experiment is to find heavy spreaders of horizontal network scans [8]. Note that all of the 120,916 destination IP addresses are not real servers.

We set θ to be 500. This means that we take any source of cardinality above 500 as a spreader or scan source. In the traffic trace, there are already 75 spreaders. We call them *innate spreaders*. We also generate 20 artificial scan sources, called *simulated spreaders*. Each simulated spreader has the cardinality of $1.1 \times \theta$. For this, we generate $1.1 \times \theta$ packets to distinct destinations, called artificial packets. We spread these packets evenly inside the original traffic trace. Let μ be the number of packets between two artificial packets. The final traffic trace includes all the packets of the original traffic trace and the artificial packets scattered evenly; there are exactly μ packets between two consecutive artificial packets.

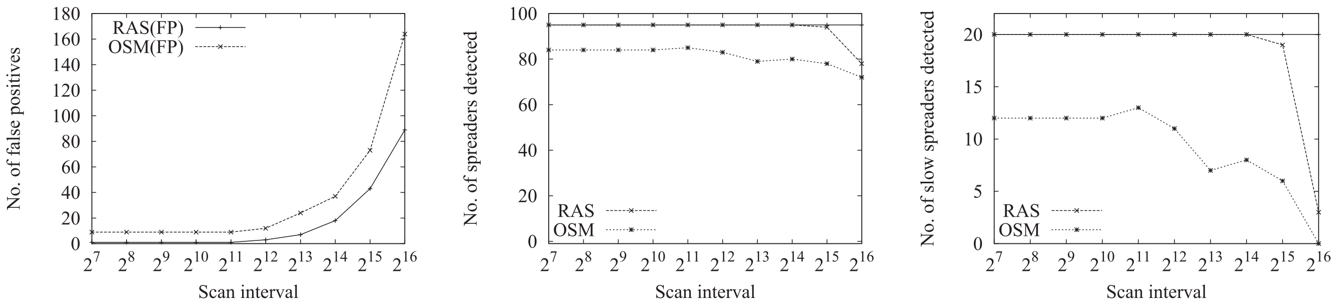


Fig. 5 Three plots show the number of false positives, the number of total spreaders detected, and the number of simulated spreaders detected, from left to right, respectively. The x-axis indicates the scan interval for the simulated spreaders, denoted by μ. The system parameters are M=512KBytes, m=256, n=2^{14}, c=9. The total number of spreaders is 95 (75 innate spreaders and 20 simulated spreaders).

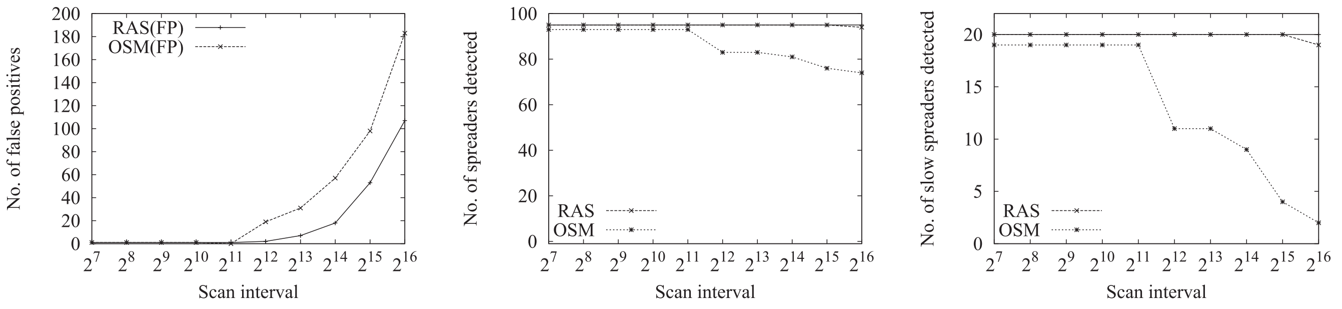


Fig. 6 Three plots show the number of false positives, the number of total spreaders detected, and the number of simulated spreaders detected, from left to right, respectively. The x-axis indicates the scan interval for the simulated spreaders, denoted by μ. The system parameters are M=1MBytes, m=256, n=2^{15}, c=9. The total number of spreaders is 95 (75 innate spreaders and 20 simulated spreaders).

This final traffic trace will be fed to OSM and RAS. On average, $\mu = 2^7$ is equal to 0.47 seconds. Therefore, if $\mu = 2^{14}$ is used for an experiment, each simulated spreader will send a packet every 60.04 seconds on average.

We always allocate the same amount of memory to OSM and RAS for fair comparison. The values of α, β, c and m are from Table 1 for RAS. We run OSM multiple times with different parameter values of m and n, with $m \times n$ unchanged. Three row indices were used, $k = 3$, as recommended an optimal value by [1]. The size of time window is 24 hours as the traffic trace was collected for 24 hours and the goal of experiments is to compare OSM and RAS in terms of detecting stealthy spreaders. If the time window had been set to a smaller value, the false positives would have decreased; however, the false negatives would have increased because the stealthy spreaders could not be detected as well as all the simulated spreaders. We choose the best result of OSM for comparison with RAS. Through the experiments, we find that OSM works best when m=256.

The actual experiments are done as follows. We execute RAS and count the number of false positives and false negatives. We run OSM with different parameter values. We pick the best one that triggers slightly more false positives than RAS. Being the best means that this OSM version detects as many spreaders as possible. Note that the number of detected spreaders decreases as the number of false positives decreases. In this sense, the comparison is advantageous to OSM as the number of false positives is larger than that of RAS. We set k to 3 for both RAS and OSM as in [1].

Figures 5~6 show the results where the memory assignment is 512KBytes and 1MBytes, respectively. We increase the memory size by doubling n. Each figure has three plots: left, middle, and right.

The left plot of Fig. 5 shows the number of false positives of RAS and OSM. Note that the plotted OSM is picked because it has detected the largest number of spreaders. The x-axis shows μ for the simulated spreaders. The middle graph shows the numbers of spreaders detected by RAS and OSM respectively. If the detection were perfect, 95 spreaders (75 innate and 20 simulated) should be detected at any x value. It is interesting that OSM fails to detect even innate spreaders. We can explain it in two ways. First, suppose a source of cardinality 501. This will have a higher chance of evading the detection if the estimated cardinality is less than 500, say 499. To the contrary, a source of cardinality 1000 will have a high chance of detection. Second, the memory reset occurs and hampers the detection. The number of memory reset remains 3 until $\mu = 2^{10}$. This number becomes 4, 7, 13, 26, 52, and 103 at $\mu = 2^{11}, 2^{12}, 2^{13}, 2^{14}, 2^{15}$, and 2^{16}.

From the middle graph of Fig. 5, we see that most innate spreaders start and finish sending packets within a short period of time. This is because the number of spreaders detected by OSM does not decrease much as μ increases. The decrease of RAS is due to the increased detection failure of simulated spreaders as μ increases. This can be confirmed in the right plot. The right plot shows that RAS detects simulated spreaders much better than OSM. Actually, RAS detects both innate and simulated spreaders better than OSM while keeping the false positives smaller than those of OSM.

We can see the same patterns in Fig. 6 where the memory size is doubled. With the memory of 1MBytes, OSM works quite good in detecting innate spreaders, as shown in the middle graph of Fig. 6. However, the simulated spreaders still cannot be detected as μ becomes large. Contrarily, RAS detects both spreaders perfectly even when μ is large.

5. Related Work

Snort is a world famous network-based intrusion detection system. To detect scan sources, snort simply keeps track of each source and the corresponding distinct destinations during a short time-window. However, this approach is not feasible for detecting stealthy spreaders [2], [9] due to the small size of SRAM.

Venkataraman et al. define a heavy distinct-hitters problem [2]. They also define a more specific problem, superspreaders, and propose two techniques for the detection. Superspreaders are heavy distinct-hitters, but they scan victims quickly. In other words, the length of time-window should be short.

Zhao et al. propose a new spread estimator for high-speed networks [1]. They introduce a lightweight data structure of two-dimensional bit table, which can reside in SRAM.

Streaming algorithms have one weakness in common; the actual flow ID is not recorded. Therefore, we need to keep the actual ID's in an extra space and look up the real values when necessary. Schweller et al. attack this problem by introducing a novel technique based on hash functions: modular hashing, IP mangling and reverse hashing [10]. Bu et al. improves the performance of reverse hashing in terms of not only accuracy but also computational overhead and memory usage [11].

Recently, Gao et al. propose to detect stealthy spreaders by using online outdegree histograms in the context of change detection [12]. Their definition of stealthy spreaders is different from ours. In their definition, stealthy spreaders are a group of sources that send scanning packets at a constant rate together. Actually, they aim to detect scans from botnets.

A sampling is a widely used technique to expand an estimate range with a limited memory space. Cao et al. propose an independent sampling with multiple Bloom filters to identify Internet hosts of a high spread value [13]. The proposed scheme removes hosts of small spread values efficiently. Kamiyama et al. propose a simple scheme to find superspreaders with a host table, which is a hash table with pointers [14]. Although these schemes work well in finding superspreaders during a short period of time, they can easily be evaded by stealthy spreaders.

SRAM is believed to be fast enough to handle network traffic from high-speed networking environments. As the size of SRAM is small, compact data structures need to be developed. Shomura et al. propose a traffic measurement scheme and argue that the prototype system can handle up to 10 Gbps traffic. However, as is mentioned in that paper[15], this would be possible when processing SYN packets only, the amount of which is far less than total traffic.

6. Conclusion

In this paper, we studied the problem of stealthy spreader detection and proposed a new streaming algorithm. The problem is caused by the negative effects of normal traffic and a total loss of accumulated information in a detection system's memory space. The proposed scheme uses the random aging, which enables the detection of stealthy spreaders. We demonstrated theoretically how the random aging elevated the detection capability. Through experiments on real Internet traffic traces, we compared the proposed scheme with the current state-of-the-art detection scheme. The proposed scheme is expected to provide practical help network/security management people to detect stealthy attacks, which has been one of the difficult problems in network security.

Acknowledgment

This work was supported by the research program 2010 of Kookmin University in Korea. This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology (2010-0008708).

References

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

[2] S. Venkatataman, D. Song, P. Gibbons, and A. Blum, "New streaming algorithms for fast detection of superspreaders," Proc. NDSS'05, Feb. 2005.

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

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

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

[6] S.M. Ross, Introduction to Probability Models, 8th ed., Elsevier, 2003.

[7] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, Introduction to Algorithms, 2nd ed., MIT Press, 2001.

[8] S. Staniford, J. Hoagland, and J. McAlerney, "Practical automated detection of stealthy portscans," J. Computer Security, vol.10, pp.105–136, 2002.

[9] M. Roesch, "Snort - lightweight intrusion detection for networks," Proc. 13th Systems Administration Conference, USENIX, 1999.

[10] R. Schweller, Z. Li, Y. Chen, Y. Gao, A. Gupta, Y. Zhang, P. Dinda,

M. Kao, and G. Memik, "Reversible sketches: Enabling monitoring and analysis over high-speed data streams," IEEE/ACM Trans. Netw., vol.15, pp.1059–1072, Oct. 2007.

[11] T. Bu, J. Cao, A. Chen, and P. Lee, "A fast and compact method for unveiling significant patterns in high speed networks," Proc. Infocom'07, pp.1893–1901, May 2007.

[12] Y. Gao, Y. Zhao, R. Schweller, S. Venkataraman, Y. Chen, D. Song, and M. Kao, "Detecting stealthy spreaders using online outdegree histograms," Proc. IEEE International Workshop on Quality of Service'07, pp.145–153, June 2007.

[13] J. Cao, Y. Jin, A. Chen, T. Bu, and Z.-L. Zhang, "Identifying high cardinality Internet hosts," Proc. IEEE Infocom'09, April 2009.

[14] N. Kamiyama, T. Mori, and R. Kawahara, "Simple and adaptive identification of superspreaders by flow sampling," Proc. IEEE Infocom'07, May 2007.

[15] Y. Shomura, Y. Watanabe, and K. Yoshida, "Analyzing the number of varieties in frequently found flow," IEICE Trans. Commun., vol.E91-B, no.6, pp.1896–1905, June 2008.

Appendix: Number of One-Bits in the Bit Table

Without loss of generality, we consider the *i*th row in the bit array. Let $A_{i,j}$ be the event that $B(i,j)$ remains zero when the bit array has ρ one-bits. Let $I_{A_{i,j}}$ be the corresponding indicator random variable.

$$P(A_{i,j}) = 1 - \frac{\rho}{n\times m}, P(A_{i,j}\cap A_{i,s}) = \left(1-\frac{\rho}{n\times m}\right)^2$$

Let Z be a random variable to represent the number of zero-bits in the row. So, $Y = m - Z$. We first compute $E(Z)$ and $E(Z^2)$.

$$Z = \sum_{j=1}^{m} I_{A_{i,j}}$$

$$E(Z) = \sum_{j=1}^{m} P(A_{i,j}) = m\times\left(1-\frac{\rho}{n\times m}\right)$$

$$E(Z^2) = E\left(\left(\sum_{j=1}^{m} I_{A_{i,j}}\right)^2\right)$$

$$= m\left(1-\frac{\rho}{n\times m}\right) + (m(m-1))\left(1-\frac{\rho}{n\times m}\right)^2$$

Note that $I^2_{A_{i,j}} = I_{A_{i,j}}$ and $I_{A_{i,s}}\times I_{A_{i,j}} = I_{A_{i,s}\cap A_{i,j}}$. Since $Y = m - Z$, we can derive $E(Y)$ and $V(Y)$ from $E(Y) = m - E(Z)$ and $V(Y) = V(Z)$.

$$E(Y) = E(m-Z) = m - E(Z) = \frac{\rho}{n}$$

$$V(Y) = E(Z^2) - E(Z)^2 = \frac{\rho}{n}\times\left(1-\frac{\rho}{n\times m}\right)$$

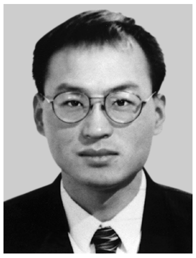
In most of the time, $\rho = \alpha\times n\times m$. By replacing ρ, we have

$$E(Y) = \alpha\times m \qquad (A\cdot 1)$$

$$V(Y) = \alpha\times m\times(1-\alpha). \qquad (A\cdot 2)$$

Alternatively, the equations can be simply derived by using the property of binomial distribution; Y obeys binomial distribution with the probability of α and the number of trials of m.

MyungKeun Yoon received the B.S. and M.S. degrees in computer science from Yonsei University, Seoul, Korea, in 1996 and 1998, respectively, and the Ph.D. degree in computer engineering from the University of Florida, Gainesville, in 2008. He is an Assistant Professor with the Department of Computer Engineering, Kookmin University, Seoul, Korea. He worked for the Korea Financial Telecommunications and Clearings Institute, Seoul, Korea, from 1998 to 2010. His research interests include computer and network security, network algorithms, and mobile networks.



Shigang Chen received his B.S. degree in computer science from University of Science and Technology of China in 1993. He received M.S. and Ph.D. degrees in computer science from University of Illinois at Urbana-Champaign in 1996 and 1999, respectively. After graduation, he had worked with Cisco Systems for three years before joining University of Florida in 2002. His research interests include network security and wireless networks. He received IEEE Communications Society Best Tutorial Paper Award in 1999 and NSF CAREER Award in 2007. He was a guest editor for ACM/Baltzer Journal of Wireless Networks (WINET) and IEEE Transactions on Vehicle Technologies. He served as a TPC co-chair for IEEE IWQoS 2009 and the Computer and Network Security Symposium of IEEE IWCCC 2006, a vice TPC chair for IEEE MASS 2005, a vice general chair for QShine 2005, a TPC co-chair for QShine 2004, and a TPC member for many conferences including IEEE ICNP, IEEE INFOCOM, IEEE ICC, IEEE Globecom, etc.