FDF: Frequency Detection-Based Filtering of Scanning Worms

Byungseung Kim and Saewoong Bahk
School of Electrical Engineering and Computer Science, INMC
Seoul National University, Seoul, Korea
Email: {kbs, sbahk}@netlab.snu.ac.kr

Hyogon Kim
Department of Computer Science and Engineering
Korea University, Seoul, Korea
Email: hyogon@korea.ac.kr

***Abstract*— In this paper, we propose a simple algorithm for detecting scanning worms with high detection rate and low false positive rate. The novelty of our algorithm is inspecting the frequency characteristic of scanning worms from a monitored network. Its low complexity allows it to be used on any network-based intrusion detection system as a real time detection module for high-speed networks. Our algorithm need not be adjusted to network status because its parameters depend on application types, which are generally and widely used in any networks such as web and P2P services. By using real traces, we evaluate the performance of our algorithm and compare it with that of SNORT. The results confirm that our algorithm outperforms SNORT with respect to detection rate and false positive rate.**

I. Introduction

Recently, worm epidemics have become a grave concern by demonstrating their formidable power to incapacitate various internet services and exhaust network resources. For instance, CodeRed, Blaster, Nimda, and SQL Slammer worms inflicted huge economic and social damages, and their mutations are still threatening the Internet environment. A distinct feature of the worms is their self-propagation behavior that is enabled by fast, automatized scanning for possible victims. They can even spread globally in just a few minutes [6], [9]. Furthermore, the technique is commonly utilized for compromising many zombie hosts from which to launch distributed denial of service (DDoS) attacks. Therefore, the detection of worm propagation in a fast and efficient manner has been a critical issue for mitigating its malignant impacts.

Most worm detection algorithms commonly look for the port-scanning behavior in which an infected host attempts to request far more new connections than a legitimate host would [3], [11], [12]. Scanning worms target several specific service ports that are known to be vulnerable to buffer-overflow. In this process, they cause high rate of failed connections. These can be verified by gathering ICMP unreachable messages in the monitored network [8] or analyzing highly anomalous traffic relative to the usual traffic distribution [4], [5], [13]. Moreover, since they use uniformly distributed IP addresses as their target hosts in random IP scanning and need responses to the scanning for finding vulnerable hosts, they expose some specific packet flows between the monitored network and the Internet. Due to these idiosyncrasies, scanning worms can be detected [1], [2].

However, there can be complications in detecting scanning worms. First, it is not easy to determine the threshold over which the suspicious behavioral patterns described above are positively identified. In fact, most detection algorithms need to tune their parameters to fit the environment they work in, such as the site and time-of-day characteristics for efficiency and accuracy. Second, some internet services that show similar behaviors with the worms are likely to cause false positives. For instance, a P2P client often behaves like a scanning worm when searching for P2P servers that have desired contents. Moreover, many web pages containing a larger number of embedded multimedia contents require many new connections. Third, recent worms such as Blaster and Agobot prefer local sequential scanning to global random scanning. Therefore, the assumption that worms perform global random scanning is not always valid.

In this paper, we propose a scanning worm detection algorithm, named Frequency Detection-based Filtering (FDF). It achieves high detection rate and low false positive rate even when the scanning traffic from worms are mingled with legitimate scanning-type flows such as P2P traffic. Moreover, it can be easily implemented on the top of any existing network based intrusion detection system (IDS) thanks to its simplicity.

The main idea of the FDF comes from the observation that scanning worms pause for a specific and characteristic period of time between individual scan attempts. TCP-based scanning worms usually transmit SYN packets at the rate prescribed by its self-propagation code. In contrast, normal TCP-based applications send SYN packets to other hosts at rather an indeterminate rate. This creates the different frequency characteristics that can be leveraged to distinguish the one from the other. The FDF extracts this frequency characteristics from just SYN arrival patterns from the monitored network, irrespective of the number of SYN packets.

II. Autocorrelation and PSD Estimation

A TCP-based worm attempts to infect hosts via SYN scanning on some accessible ports. Since the scanning logic is programmed as a loop in its self-propagation code, the worm periodically generates SYN packets towards victim hosts. This is the worm frequency characteristics. For instance, CodeRedII, forces an infected host to create 300 threads, and each thread periodically runs a scanning process with the inter-scan sleep period of 100 msec[1] [14]. That is, a host infected by CodeRedII scans at the rate of 10 (packets/sec) per thread that exhibits the frequency characteristic of 10 Hz. Sleep instructions deciding the frequency characteristics are also observed in the disassembled codes of other scanning worms such as Blaster and Sasser [15], [16].

A. *Autocorrelation and PSD in Real Traces*

Assume that legitimate SYN arrivals follow the Poisson distribution. Let X_n and Y_n be the random variables that represent the numbers of SYN arrivals from legitimate TCP sessions and a scanning worm in the interval (t_{n-1}, t_n), respectively, where $t_{n-1} = t_n - T$ and T is the sampling interval for counting the number of SYN arrivals. Then,

$$P(X_n = k\ ) = \frac{(\lambda T)^k e^{-\lambda T}}{k!} \qquad \text{for } k = 0, 1, 2, \dots\ ,$$

$$P(Y_n = A_w) = \begin{cases} 1 & \text{if } n = kN_w, \\ 0 & \text{if } n \neq kN_w, \end{cases}$$

where A_w is the number of active threads created by an infected host, and N_w is the scanning period of the worm. The autocorrelations of X_n and Y_n are given by

$$\begin{aligned} R_{XX}(m) &= (\lambda T)^2 + \lambda T \delta(m), \\ R_{YY}(m) &= \frac{1}{N - |m|} \sum_{n=0}^{N-m-1} Y_{n+m} Y_n \\ &= \frac{{A_w}^2}{N_w} \delta(m - kN_w). \end{aligned}$$

Letting $Z_n = X_n + Y_n$, we obtain

$$\begin{aligned} R_{ZZ}(m) =& (\lambda T)^2 + \frac{2\lambda T A_w}{N_w} + \lambda T \cdot \delta(m) \\ &+ \sum_{k=-\infty}^{\infty} \frac{{A_w}^2}{N_w} \delta(m - kN_w). \end{aligned} \tag{1}$$

We applied the autocorrelation estimate to real traces. The traces used in our experiments have been collected by TCPDUMP at the gateway router between a university and the Internet for 25 hours.[2] A summary of our traces is given in Table I. The 'Outgoing' in the table represents unidrectional packet departures from the university to the Internet. '03_out denotes the outgoing of our trace for the year 2003, and '04_out denote the outgoing of our trace for the year 2004. To check whether any known worms exist in our traces, we applied the signatures of SNORT2.0 before experiments, and we identified two hosts with CodeRedII in the 2003 trace but none in the 2004 trace. In fact, there were several hosts suspected of some type of infection in the 2004 trace. But we could not identify the worm(s) by signature matching because there were no successful setup of TCP connections over which to transport exploit codes.

[1]Actually, all the threads are not always active due to the limited number of sockets, limited network resources, and so on. In the inspection of our Internet traces, just 8∼10 SYN arrivals from a CodeRedII were observed every 100 msec. That is, the number of running thread was about 10.

[2]We picked this university from among those using the KREN (Korea Education Network). The traces are for the years 2003 and 2004, for the selected school.

TABLE I
A SUMMARY OF THE CAPTURED TRACES BY TCPDUMP

Trace	IP Block	Capturing Time	Avg. BW
'03_out	/19 network /20 network	11:37, July 3, 2003 ∼ 13:00, July 4, 2003	Outgoing 45Mbps
'04_out	/19 network /20 network	23:15, July 28, 2004 ∼ 23:42, July 29, 2004	Outgoing 45Mbps

In the measurement of autocorrelation and PSD, each sample represents the number of SYN arrivals for a time bin of 10 msec, and one sample set consists of 10,000 samples. We obtained 912 sample sets in the 2003 trace and 880 sample sets in the 2004 trace. Fig. 1(a) shows the example results for the 15th set in the 2003 trace. To investigate the effect of CodeRedII in the 2003 trace, we perform the autocorrelation estimates before and after excluding SYN arrivals of the two infected hosts from the trace. The results are shown in Figs. 1(c) and (b) respectively. Clearly, the autocorrelation provides a visible indication on the existence of scanning worms. Fig. 1(d) shows a PSD estimate of the time series. We notice that there are SYN arrivals from the worm with the frequency of 10 (=1/0.1)Hz that bears out the periodicity of 10 sample lag (=0.01×10 sec) of Fig. 1(c).

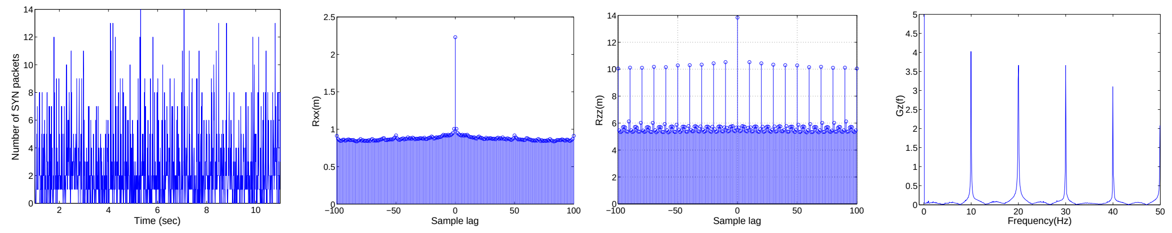
In essence, the autocorrelation and PSD estimates show that the frequency characteristic of SYN arrivals from legitimate hosts spreads out all over the frequency band, whereas that from an infected host does not. Thus they can help to identify unknown or slow-scanning worms.

B. *Discussion*

In obtaining the specific frequency characteristics as shown in Figs. 1(c) and (d), there are several caveats. First, a significant fraction of SYN transmssions from an infected host should be captured at the monitor for better detection accuracy. This means that the measurement against outgoing SYNs is likely to generate stronger results than against incoming SYN arrivals. This is because usually, the incoming scanning SYN packets from the Internet spread out over many sites.

Second, the sampling period T for obtaining a sample set should be less than the scanning period T_w $(= N_w \cdot T)$ of a worm. According to the Nyquist sampling theorem [17], the sampling rate should be at least $1/T_w \cdot 2$ Hz. We enforce it in the estimates of autocorrelation and PSD given the target frequency characteristic of a worm is lower than or equal to $1/T_w$ Hz. Since most operating systems manage the interrupt of a packet arrival within a few micro seconds, there is no difficulty in setting the sampling rate higher. In our experiments, we set the sampling period T at 10 msec that enables us to detect the worm frequency of up to 50 Hz.

Lastly, we have to consider the processing time of the autocorrelation and PSD estimates for real-time detection. Since the observable frequency is proportional to the sample lag m and the sampling interval T, the autocorrelation with



(a) Time series

(b) Autocorrelation without the infected hosts

(c) Autocorrelation with the infected hosts

(d) PSD estimate

Fig. 1. Time series, autocorrelation and PDS estimates of sample sets in the 2003 trace (outgoing).

m lags needs to be computed $m \cdot n$ times where n is the number of samples. To obtain the PSD estimate, we need to perform the Discrete Fourier Transform (DFT) for the obtained m lags of the autocorrelation. Accordingly, the autocorrelation estimate runs in $O(mn)$ and the PSD estimate in $O(m \log m)$. In our experiments, we set m at 100, n at $10,000$, and T at 0.01. Then, we can detect the worm having scanning frequency of 1 $(= 100 \times 0.01)$Hz through 50 $(= 1/0.01 \times 1/2)$Hz. From these parameters, the autocorrelation estimate and the PSD estimate are just computed $100 \times 10,000$ times and $100 \times 10,000 + 100 \log 100$ times, respectively.

III. Frequency Detection Based Filtering

To distinguish infected hosts from legitimate hosts, we perform the autocorrelation estimate of SYN arrivals on individual host basis. The complexity of the autocorrelation estimate with m lags and n samples is $O(mn)$, but with N hosts to inspect it is $O(Nmn)$. To cope with the high complexity, here we propose a $O(1)$ method for the FDF.

A. FDF Algorithm

Fig. 2 shows the pseudo code of the FDF that has the following features:

- Time slot T_s - This is determined according to the autocorrelation estimate. For example, if the autocorrelation estimate is to detect the SYN scanning at 10Hz, we set the time slot length to 0.1 sec.
- Hash table entry - A distinctive characteristic of scanning worms is that they scan a specific port known to be vulnerable. That is, SYN packets from an infected host have one source IP (SIP), many destination IPs (DIPs), many source Ports (SPs) and a single destination Port (DP). Therefore, we define a "flow" as SYN arrivals with the same SIP and DP pair. The FDF creates a hash table with the key consisting of the SIP and DP, and updates it according to the arrival of a new DIP. An entry of the hash table consists of a duration counter n_d, which counts the number of consecutive time slots of observing a flow, the starting time slot t_s for the flow, the latest updated time slot t_l, and SYN counter n_s indicating the number of SYN arrivals during the current time slot.
- Duration threshold D_{th} - If the worm scanning lasts more than D_{th}, the FDF raises an alarm.
- Update margin U_m - FDF keeps counting even if there is no scanning activity for less than U_m slots. This handles the heavily local scanning worms with minimal global scanning component. Unless they are properly handled, they could cause false negatives.

```
for each SYN packet:
    if Hash(SIP,DP)
        if $t - t_l < U_m$
            Update_ table_ entry(SIP,DP)
                $n_d++$ ; $t_l = t$ ;
        else
            Initialize_table_entry(SIP,DP)
                $t_s = t_l = t$ ; $n_d = 1$ ;
    else
        Initialize_table_entry(SIP,DP)
            $t_s = t_l = t$ ; $n_d = 1$ ;
    if $n_d > D_{th}$
        Alarm(SIP,DP)
```

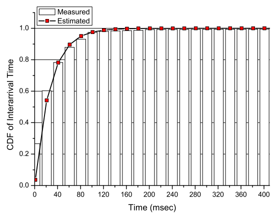
Fig. 2. Pseudo code of FDF algorithm.

The FDF detection system has three stages: pre-detection, detection, and defense. In the pre-detection stage, the system determines whether the frequency characteristic of a scanning worm appears through the autocorrelation estimate. If the frequency characteristic is detected, the FDF locates infected hosts in the detection stage. Finally, in the defense stage, the information about the infected hosts is handed over to the defense system and the infected hosts are isolated from the network.

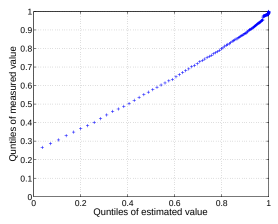
B. Behavior of SYN Arrivals from Legitimate Hosts

For optimal operation of the FDF system, we need to obtain the system parameters that allow us to tell the SYN arrivals caused by scanning worms from those by legitimate hosts. To do so, here we investigate the properties of the SYN arrivals from web clients and P2P clients that are the two predominating applications today.

These days P2P traffic accounts for a considerable share of the Internet traffic. Interestingly, hybrid P2P systems perform SYN scanning like a worm. In such systems the P2P client



(a) The measured and estimated CDF of SYN interarrival times from a P2P



(b) Q-Q plot

Fig. 3. Distribution of SYN interarrival time of P2P traffic.

sends SYN packets to the servers on the list obtained from the P2P master server to check their liveness and round-trip time (RTT) before searching files or refreshing the server list. So, it may well cause false positives. With our Internet traces, we attempted estimating the SYN interarrival time distribution for a P2P client when it searches P2P servers. Our traces contain many popular P2P traffic trace produced by such applications as e-donkey, Soribada, KaZaa, V-share, and File-Guri (some are local versions). Fig. 3(a) shows the CDF of SYN interarrival time of a V-share client fitted against the exponential distribution with $\lambda = 37$ (packets/sec). Fig. 3(b) is the QQ-plot which indicates that the SYN interarrival time of V-share is approximately exponentially distributed. Other P2P clients also exhibit similar characteristics with various mean rates of 10 through 60 (packets/sec) during each SYN burst.

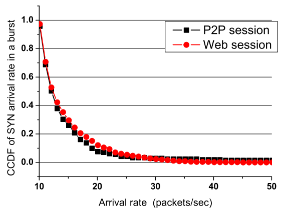
We also observe from our traces that when a web client issues multiple http requests for embedded objects on a web page, it generates about 5∼80 outgoing SYN packets in a burst. This type of burst SYN arrivals can cause false positives in the FDF system. In this paper, however, we do not address the problem because there are many prior work on the subject [10] and because P2P clients cause much more false positives than web clients.

Now we investigate the burst duration and the SYN arrival rate of P2P and web clients in a SYN burst. We represent the burst event of a flow as SYN packets that arrive at the rate of more than 10 (packets/sec) when it is active and it have the idle period of less than 1 (sec) since the human interaction causing the idle period commonly takes more than 1 (sec). Fig. 4 shows that the SYN arrival rate in a burst is smaller than 60 (packets/sec), and the burst most likely lasts less than 8 (sec).

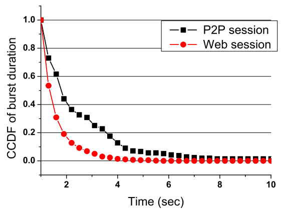
C. Parameter Adjustment

According to the observation of legitimate SYN arrivals described above, we adjust the parameters of the FDF: time slot length, duration threshold, and update margin.

1) Time slot length: The slot length, T_s, is determined by the autocorrelation estimate or the PSD estimate. As shown in Fig. 1(a), the autocorrelation estimate of legitimate SYN arrivals exhibits some fluctuation. However, we can clearly



(a) SYN arrival rate in a burst



(b) Burst duration

Fig. 4. Complementary CDF (CCDF) of SYN arrival rate in a burst and the burst duration for P2P and Web clients.

differentiate the impulse train of a scanning worm from the reference value of legitimate traffic. Let's take the reference value to be the minimum of $R_{ZZ}(m)$, and denote it by $R_{ZZ}^{\min}$. Considering error and noise denoted by $\epsilon(m)$, we can represent $H(m)$ as a likelihood function for decision as

$$H(m) = \frac{R_{ZZ}(m) - R_{ZZ}^{\min}}{R_{ZZ}^{\min}} = \begin{cases} \dfrac{\frac{A_w{}^2}{N_w} + \epsilon(m)}{(\lambda T)^2 + \frac{2\lambda T A_w}{N_w}} & \text{if } m = m^*, \\ \dfrac{\epsilon(m)}{(\lambda T)^2 + \frac{2\lambda T A_w}{N_w}} & \text{if } m \neq m^*, \end{cases} \tag{2}$$

where m^* is the sample lag at which the impulse train appears on the autocorrelation estimate.

In this equation, we have to find m^* satisfying $H(m^*) > h$ with small error where h represents a fluctuation threshold. λ is large enough to ignore the impact of $\epsilon(m)$ since λ is the total arrival rate of legitimate SYN arrivals. Accordingly, $H(m)$ for $m \neq m^*$ is kept small in comparison with $H(m^*)$. In our experiments, the maximum value of $H(m)$ for $m \neq m^*$ was about 0.2. Thus, we set a fluctuation threshold, h, at 0.3 by adding some safety margin to reduce the error possibility. Our measured value of h can be applied to other networks because it does not depend on the monitored network but the application services. Considering the variation in h, we group all the m^*'s according to the range of autocorrelation value. Intuitively, if we have a group of $\{m_0^*, m_1^*, m_2^*, ...\}$ that have similar autocorrelation values within the variation h, we obtain the frequency characteristic that has the period α and satisfies $m_k^* \in \{m | m = \alpha k + m_0^*;\ k = 0, 1, 2, 3, \ldots\}$. If there are too many groups, we can use the PSD estimate as a tool for investigating the frequency characteristics. The FDF sets the time slot length to the maximum among these obtained periods, α's. This is because SYN arrivals that have some other short interarrival times can be continuously observed within the maximum period.

2) Duration threshold: The duration threshold, D_{th}, is the minimum period for continuous monitoring. This is needed to differentiate SYN arrivals generated by a worm from those by legitimate traffic. If the threshold is too low, the FDF would brand many legitimate hosts as infected. Namely, if there are many P2P clients that search P2P servers, the FDF would generate many false positives. In contrast, if we set it too high, there exists a possibility of false negatives. Therefore, we need to find the optimal threshold. To do so, we take advantage of the features of legitimate SYN arrivals analyzed in Section III-B. Since it is likely that P2P clients cause more false positives than web clients, we focus on P2P clients.

Since SYN arrivals from a P2P client can be approximated by the Poisson distribution, we can obtain the probability that the number of observed SYN arrivals from the P2P client for T_s is greater than n as

$$P_a = p\,[k > n] = 1 - \sum_{k=0}^{n} \frac{e^{-\lambda_b T_s}(\lambda_b T_s)^k}{k!}, \qquad (3)$$

where λ_b is the average SYN arrival rate in a burst. In our experiments, we set λ_b to 60 packets/sec that gives the CCDF probability of less than 0.01 in Fig. 4(a).

Since P2P and web clients keep generating SYN packets for a relatively short duration as shown in Fig. 4(b), they do not have to be monitored for $max\_duration$. In contrast, a worm performs scanning for a somewhat long time, so it needs to be monitored for a $min\_duration$ at least. As the FDF operates in the time slotted manner, we can denote the $max\_duration$ and the $min\_duration$ by d_{max} and d_{min} in the unit of a time slot, respectively. Accordingly, for a given target probability P_f of false positives, we can obtain an optimum duration threshold D_{th} as the following.

$$D_{th} = \min\{d | {P_a}^{d} \le P_f\} \;\; \text{for} \;\; d_{min} \le d \le d_{max} \;. \qquad (4)$$

In our experiments, we set the $max\_duration$ and the $min\_duration$ to 8 sec and to 0.5 sec, respectively, which has the P_f of 0.01 in Fig. 4(b). That is, if T_s is 0.1, d_{min} is 5 (=0.5/0.1) and d_{max} is 80 (=8/0.1). We can reduce false positives of the FDF by increasing d_{max}. However, the larger d_{max} is, the larger the detection time of the FDF is.

3) Update margin: Since the performance of the FDF is affected by the network latency and the local scanning ratio of the scanning worm, its deployment locale needs to be as close to the monitored hosts as possible. In practice, it is likely to be located at the network boundary with other security systems for efficient network management. Therefore, we need to allow some update margin, U_m, for the error that can incurred by the latency and the local scanning. For example, if a worm periodically performs local scanning for six slots, the FDF needs to ignore the six slots or more (update margin) and keep updating. In our experiments, we set the update margin at six. This value is large enough to cover our /19 and /20 networks because most worms perform global scanning as well as local scanning to /16 and /8 networks. To keep the rate of false positives below a certain threshold, d_{min} and d_{max} should be scaled by the update margin.

IV. Performance Evaluation

For performance comparison, we apply the FDF to our two traces in Table I and compare the results with those from SNORT, a well-known IDS tool.

A. FDF against Real-life Traces

From the analysis of our traces with the parameter adjustment in Section III, we obtained the FDF parameters as shown in Table II and used them for evaluating the FDF.

TABLE II
A summary of obtained FDF parameters for real traces

Parameter set	T_s	λ_b	d_{min}	d_{max}	U_m	P_f
’03_out	0.1	60	30	480	6	0.01
’04_out	0.6	60	5	80	6	0.01

To investigate how many scanning worms are in the traces, we applied SNORT2.0 that executes the worm signature detection and the spp_portscan detection module. Then, we manually compared the detected flows with those obtained by the signature behaviors given in the online worm libraries [7]. As a result, we found two hosts infected by CodeRedII from ’03_out, six infected hosts from ’04_out. To identify any false positives, we examined the packet payload and DIPs for each detected flow. We found several false positives that are created by http, SMTP, and P2P. In the case of the detected SMTP flow of ’04_out, we noticed that there are many HTML contents like spam mails and all the DIPs correspond to legitimate SMTP servers. In the same manner, it was easy to verify the remaining false positives by inspecting the port numbers and the delivered contents.

Table III summarizes the results of our experiments. Except for the CodeRedII, most of the detected scanning flows exhibited sequential scanning behavior. As they failed to establish connections, we could not determine what kind of worms they were. Assuming that only a worm performs SYN scanning to infect other hosts, we inferred the type of each worm by investigating the target port list and showed it on the second column [7].

B. Comparison with SNORT

To compare the performance of the FDF with that of SNORT, we applied the spp_portscan module of SNORT2.0 to ’04_out trace. The spp_portscan module counts new connection requests from a host for a specific interval. If the count is greater than the threshold, the module regards the host as a scanner or a worm. For instance, if we set the two parameters of the threshold and the interval to 90 and 30, which are denoted by 90/30, the module detects the host that attempts to make connections more than 90 times within 30 seconds. Because there is no reference concerning the parameter set of spp_portscan, we varied the parameters from extremely low to high and found the optimal values at which the number of false positives is minimal without having false negatives. In our experiments, the result of ’04_out trace shows a same optimal value for a duration of longer than 60 seconds. Considering its detection time, we set the interval of the spp_portscan to $x/60$ where x is the threshold value of the number of SYN packets observed in 60 seconds.

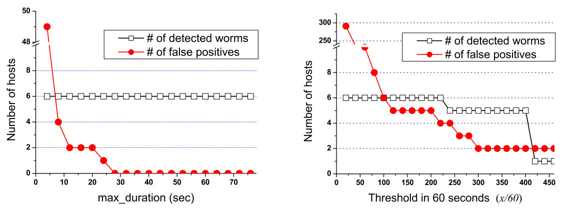
In Fig. 5, we represent the relation between the numbers of false positives and detected scanning worms. In the case of the

TABLE III

A SUMMARY OF THE RESULTS OF FDF FOR REAL-LIFE TRACES

	Detected Port	# of Flows	AvgRate(#/sec)	AvgDuration(sec)	Specification
'03_out	80(CodeRedII)	2	73.7	6072.4	Random Port Scanning
	8404(v-share)	3	45.6	192.0	False Positive(P2P Scan)
'04_out	1025(W32.Keco)	1	35.0	11664.0	/24 Sequential Scanning(1KB payload)
	3140(Optix)	5	19.7	5275.8	/24 Sequential Scanning(1KB payload)
	80(http)	1	3.05	96.0	False Positive
	25(SMTP)	1	2.3	144.0	Spam Mail

FDF, the number of detected scanning worms is fixed at six even if we increase the $max\_duration$. In the case of SNORT, both the numbers of false positives and detected scanning worms decrease as the threshold increases. Consequently, the spp_portscan generates more false negatives with the increase of the threshold while false positives are reduced. In essence, the FDF is more effective in detecting scanning worms than the spp_portscan of SNORT with less false positives.



(a) FDF (b) SNORT

Fig. 5. Performance comparison of false positives and detected scanning worms in '04_out trace.

V. DISCUSSIONS

The FDF could be used for detecting periodic attacks other than scanning worms. For example, since most SYN flooding attacks are automated by scripts or worms, they expose some frequency characteristics that can be detected by the FDF. Such an example is the Blaster worm that is programmed to execute the SYN flooding attack sending SYN packets to a target host every 20 msec [16].

Except for updating and checking the hash table entry for every SYN arrivals, the FDF do not require any other per-packet processing. Therefore, its operational complexity is $O(1)$ on each SYN arrival. As a result, it can be easily added to existing IDS without loss of performance so run on a high speed network.

Finally, one might ask what happens if an attacker programs a worm self-propagation code to have a non-deterministic sleep instruction? In this case, it is likely that the scanning interval will have some probability distribution with possibly minimum and maximum values. This still would give some frequency characteristics, but the accuracy of detection could get lower. We will address this issue in our future work.

VI. CONCLUSION

In this paper, we presented a simple and effective scanning worm detection algorithm, FDF. The FDF algorithm exploits the frequency characteristic of worm traffic, which is created by the deterministic iteration of worm codes. Notice that the frequency characteristic is independent of the number of SYN packets. So, the FDF can detect slow scanning worms as well as fast scanning worms. This contrasts with existing threshold-based detection systems that have difficulty in detecting slow scanning worms because a small number of SYN packets generated by them do not cause visible traffic anomaly. Moreover, the FDF has low implementation complexity and the system parameters are independent on the network size and time-of-day. These features make it a promising method that can be deployed in any IDS systems and to run on a high-speed link.

ACKNOWLEDGEMENTS

This research was supported by University IT Research Center Project and NRL program of KISTEP

REFERENCES

[1] C. Zou, L. Gao, W. Gong, and D. Towsley, *Monitoring and Early Warning for Internet Worms*, the 10th ACM CCS, 2003.

[2] G. Gu, M. Sharif, X. Qin, D. Dagon, W. Lee, and G. Riley, *Worm Detection, Early Warning and Response Based on Local Victim Information*, the 20th Annual Computer Security Applications Conference.

[3] M. Williamson, *Throttling Viruses: Restricting Propagation to Defeat Malicous Mobile Code*, June, 2002.

[4] J. Jung, V. Paxon, W. Berger, and H. Balakrishnan *Fast Portscan Detection Using Sequential Hypothesis Testing*, IEEE Security and Privacy, 2004.

[5] C. Leckie and R. Kotagiri, *A Probabilisitic Approach to Detecting Network Scans*, IEEE NOMS02.

[6] S. Staniford, V. Paxson, and N. Weaver, *How to 0wn the Internet in your Spare Time*, the 11th USENIX Security Symposium, 2002.

[7] doshelp, *trojanports*, http://www.doshelp.com/trojanports.htm.

[8] G. Bakos and V. Berk, *Early Detection of Internet Worm Activity by Metering ICMP Destination Unreachable Activity*, SPIE, 2002.

[9] Z. Chen, L. Gao, and K. Kwiat, *Modeling the Spread of Active Worms*, IEEE INFOCOM, 2003.

[10] M. E. Crovella, *Self-Similarity in World Wide Web Traffic: Evidence and Possible Causes*, IEEE/ACM Transactions on Networking, Vol. 5, No.6, December 1997.

[11] M. Roesch, *Snort: Lightweight intrusion detection for networks*, LISA-99, 1999.

[12] Herberlein, L.T., G. Dias, K. Levitt, B. Mukherjee,J. Wood, And D. Wolber, *A network security monitor*, Symposium on Research in Security and Privacy, 1990.

[13] S. Staniford, J. A. Hoagland, J. M. McAlerney, *Practival Automated Detection of Stealthy Portscans*, Journal of Computer Security, Volume 10, Issue 1-2, 2002.

[14] CAIDA, *CAIDA Analysis of Code-Red*, http://www.caida.org/analysis/security/code-red/#crii.

[15] eEye Digital Security, *Sasser worm analysis*, http://www.eeye.com/html/research/advisories/AD20040501.html.

[16] eEye Digital Security, *Blaster worm analysis*, http://www.eeye.com/html/research/advisories/AL20030811.html.

[17] V. Oppenheim and W. Schafer, *Discrete-Time Signal Processing*, Prentice Hall.