

# An Internet-Worm Early Warning System

Shigang Chen      Sanjay Ranka

Department of Computer & Information Science & Engineering

University of Florida, Gainesville, FL 32611, USA

{sgchen, ranka}@cise.ufl.edu

**Abstract**—We propose an Internet-worm early warning system, which integrates a set of novel techniques that automatically detect the concerted scan activity of an on-going worm attack. It is able to issue warning at the early stage of worm propagation and to provide necessary information for security analysts to control the damage. The system monitors a “used” address space. Unlike the traditional approach that keeps track of SYN packets, it relies on RESET packets to find the scan sources, which has greater accuracy and less overhead. The system is resilient to anti-monitor measures. Particularly, a sophisticated protocol is designed to distinguish faked scan sources from real scan sources. We provide an analytical study on the properties and effectiveness of this early warning system, and back up our claims by numerical results.

## I. INTRODUCTION

A worm exploits software security loopholes (often the buffer-overflow problem) to compromise vulnerable systems, which are then used to scan the Internet for more victims. As more and more machines are compromised, more and more copies of the worm are working together to reproduce themselves. An explosive epidemic is therefore developed across the Internet. Most recent research on Internet worms concentrates on propagation modeling [1]–[4]. The defense against worms is still an open problem. Moore et al. has recently studied the effectiveness of worm containment technologies (*address blacklisting and content filtering*) and concluded that such systems must react in a matter of minutes and interdict nearly all Internet paths in order to be successful [4]. Williamson proposed to modify the network stack so that the rate of connection requests to “new” hosts is bounded [5]. The idea is to slow down the worm propagation by bounding the maximum scanning rate of any worm-infected host. The main problem is that the effectiveness of this approach requires universal deployment.

Chen, Gao, and Kwiat proposed a sophisticated worm propagation model (called AAWP [6]) based on discrete times. In the same work, the model is applied to monitor, detect, and defend against the spread of worms under a rather simplified setup, where a set of unused addresses are monitored and a connection made to those addresses triggers a worm alert. There are two problems with this approach. First, as noted in the paper, the attackers can easily overwhelm such a system with false positives by sending packets to those addresses. Second, to achieve good response time, the number of “unused addresses” has to be large, but addresses are scarce resources in the IPv4 world, and only few has the privilege of establishing such a system. A monitor/detection system based on

“used addresses” will be much more attractive. It allows more institutes or commercial companies to participate in the quest of defeating Internet worms.

An early warning system is essential in fighting against natural disasters such hurricanes, floods, wildfires, etc. Even for less-predictable tornados or earthquakes, a just-in-time warning can be invaluable in saving lives and limiting damages. Similarly, in the Internet world, a worm early warning system is extremely important due to the enormous harm that a worm can potentially cause [1]. Such a system is also practically feasible, given the worm’s unique behavioral characteristics that have been well established, thanks to the flourish of recent research results in worm modeling.

In this paper, we propose WEW, an Internet-worm early warning system, which integrates a set of novel techniques that automatically detect the concerted scan activity of an on-going worm attack. Our contributions are listed below.

- The proposed worm-detection methods do not require unused address space. Unlike the traditional approach that keeps track of TCP SYN, we rely on TCP RESET packets to find the scan sources, which has greater accuracy and less overhead.
- We propose an anti-spoof protocol that filters out the false scan sources and identify the possible worm-infected hosts. The protocol covers various cases that may happen according to the TCP protocol. It remains stateless by using cookies, which makes it robust against denial-of-service (DoS) attacks.
- We provide an analytical study of the system and propose a new performance metric, *system sensitivity*, to capture the responsiveness of an early warning system in reporting an on-going worm. We also demonstrate the numerical results of the system to support our claims.

## II. INTERNET-WORM EARLY WARNING SYSTEM

### A. Normal Activity vs. Worm Activity

We study the behavioral differences between a normal user and a worm-infected host for common applications such as web browsing.

- A normal user accesses a server by a domain name. The domain name is resolved for an IP address via DNS (Domain Name System). If the domain name cannot be resolved, no TCP connection will be made. On the other hand, a worm attempts TCP connections to random addresses no matter whether these addresses are alive or not, which results in a large number of connection failures.

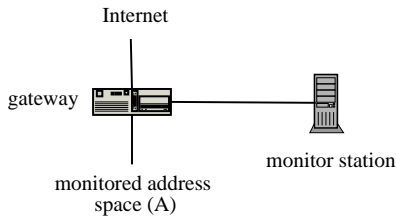


Fig. 1. WEW: Sampling the Internet worm activity

- Comparing with worm-infected hosts that scans hundreds of addresses per second, normal users connect to different servers at much slower rates due to manual operations and reading time.

- A user typically has a favorite server list. Those servers are visited most often and they are known to be up most of the time.

In summary, a worm-infected host will generate a persistent stream of failed connections, often at a high rate, while a normal user generates failed connections occasionally, at a much slower rate that does not persist. By observing the behavioral differences, we can distinguish normal users from worm-infected hosts.

We focus on worms that spread by randomly scanning the Internet. Worms that perform targeted scanning [1], which may infect the Internet in seconds, are beyond the scope of this paper.

### B. Monitoring Scan Sources

The system architecture of our Internet-worm early warning system (WEW) is shown in Fig. 1. It samples the Internet scan activities by monitoring a portion of the IPv4 address space. With the assistance of the gateway, the monitor detects potential worm outbreak by analyzing the pattern of increase in scan sources and comparing their similarity.

Let  $A$  be the monitored address space, which can be a used address space. A gateway separates  $A$  from the Internet. The primary task of the monitor station is to profile all scan sources from the Internet. One naive approach is to keep track of the inbound TCP SYN packets. If the number of SYN packets from an external host exceeds a threshold value within a period time, the host is thought to be scanning. Although this approach is commonly used in commercial intrusion-detection systems, it is however neither accurate nor efficient. The number of TCP connections through a major router can be hundreds of thousands per second. Monitoring all TCP SYN incurs significant overhead because it spends computation/storage resources on all TCP connection attempts, even though only a small fraction may come from scan sources.

A better approach is to monitor TCP RESET packets that indicate failed connection attempts. A connection fails if the destination host does not exist or the destination port is not open. An ICMP host-unreachable packet is returned if a SYN packet is sent to a non-existing host; a TCP RESET packet is returned if a SYN packet is sent to an existing host with the

targeted port closed. Our experiment shows that random scan for web servers fails over 99.6% times. The percentage will be even higher for less popular services. Hence, Monitoring TCP RESET and ICMP host-unreachable catches a majority of the scan traffic. If  $A$  is densely populated, then monitoring RESET alone will suffice. We address the problem of sparsely populated  $A$  in Section II-E.

As we argued previously, a normal user does not persistently generate a large volume of failed connections. Note that connection failures caused by network congestion or server downtime do not generate TCP RESET and will not be counted. Therefore, by monitoring RESET packets, we can set scan sources apart from normal users.

### C. Identifying False Scan Sources

An adversary may defeat the above system by generating false scan sources to cover its activity. Moreover, if the number of false sources is overwhelmingly large, it can degrade the performance of the monitor system or even constitute a denial-of-server attack.

As an example, a hostile host may cause WEW to generate a false-positive by simulating a worm attack. It starts with connections to random destination addresses from a single forged source address. As time passes, it makes random connections from more and more forged source addresses with the number growing exponentially. The system that monitors failed connections will detect a worm-like increase in scan sources, and consequently raise a false alarm.

We propose the following solution to the above problem. Consider a connection attempt from an Internet host to a host in  $A$ . Let  $addr\_I$  and  $port\_I$  be the address and the port of the Internet host, respectively. Let  $addr\_A$  and  $port\_A$  be the address and the port of the host in  $A$ , respectively. Let  $ackNum$  and  $seqNum$  be the acknowledgement number and the sequence number of a TCP segment, respectively. Let  $key$  be a secret known only by the gateway. To distinguish different messages of the same type, we use a subscribe number after the type.

When the gateway receives a TCP RESET<sub>1</sub> packet from  $A$  to the Internet, it neither notifies the monitor station immediately nor forwards the packet. Instead, it constructs a TCP SYN/ACK<sub>1</sub> packet with  $addr\_I$ ,  $addr\_A$ ,  $port\_I$ ,  $port\_A$ , and  $ackNum$  copied from the RESET<sub>1</sub> packet, with  $seqNum$  being a keyed hash,  $hash(addr\_I | addr\_A | port\_I | port\_A, key)$ , and with the other header fields properly set, where “|” is the concatenation operator. The keyed hash (using SHA1 or MD5) serves as the authentication code for the concatenation of the source/destination address/port. The above constructed SYN/ACK<sub>1</sub> packet is sent out, and the RESET<sub>1</sub> packet is dropped by the gateway. The possible outcomes from the SYN/ACK<sub>1</sub> are listed below. The first three cases assume the RESET<sub>1</sub> packet is caused by a SYN<sub>1</sub> packet from the Internet.

Case 1: If the initial SYN<sub>1</sub> packet carries a forged source address that does not exist, then SYN/ACK<sub>1</sub> will result in an ICMP host-unreachable packet being returned.

Case 2: If the initial SYN<sub>1</sub> packet carries a forged source address that exists, then SYN/ACK<sub>1</sub> will result in a RESET<sub>2</sub> packet being returned.

Case 3: If the initial SYN<sub>1</sub> packet is from a real source, then SYN/ACK<sub>1</sub> will result in an ACK<sub>2</sub> packet being returned to complete the connection.

Case 4: If the dropped RESET<sub>1</sub> was not in response to a SYN packet, then SYN/ACK<sub>1</sub> is not expected by its receiver and thus a RESET<sub>2</sub> packet will be returned.

For Case 3, the gateway should verify the keyed hash, which is now in the acknowledge field. If the verification is successful, it resets the connection and *reports a failed-connection attempt to the monitor station*. This is the only case that indicates a possible real scan source.

For Case 1, the result is the same as if RESET<sub>1</sub> was forwarded (instead of being dropped). The gateway does not do anything. The gateway cannot distinguish Case 2 and Case 4 because it receives the same RESET<sub>2</sub> packet in both cases. Hence it treats them in the same way by sending back a RESET<sub>3</sub> after verifying the keyed hash. Most fields in RESET<sub>3</sub> can be copied from RESET<sub>2</sub>.

The hash verification is performed as follows. For each ACK or RESET packet received from the Internet, the gateway checks if *ackNum* equals *hash(srcAddr | dstAddr | srcPort | dstPort, key)*, where *srcAddr*, *dstAddr*, *srcPort*, and *dstPort* are the source address, destination address, source port, and destination port of the packet, respectively. The chance for an arbitrary packet to satisfy this condition is  $1/2^{32}$ , which is negligibly small. Even if that happens, there is no serious damage other than that the connection to be completed by ACK is mistakenly terminated.

If the hash verification fails, i.e.,  $ackNum \neq hash(srcAddr | dstAddr | srcPort | dstPort, key)$ , then the packet is forwarded normally. Otherwise, the packet must be the response to a previously constructed SYN/ACK<sub>1</sub>. The gateway does one of the following.

1. If the packet is an ACK<sub>2</sub> (Case 3), the gateway replies back a RESET<sub>3</sub> to terminate the connection, reports a failed-connection attempt to the monitor station, and drops the ACK<sub>2</sub>.

2. If the packet is a RESET<sub>2</sub> (Case 2 or 4), the gateway replies back a RESET<sub>3</sub> and drops the RESET<sub>2</sub>.

The handling of the four cases is illustrated in Fig. 2.

#### D. Blocking Persistent Scan Sources and Issuing Alert

Based on the connection-failure reports from the gateway, the monitor station generates a list of possible scan sources, denoted as  $\Omega$ , consisting of external addresses  $x$  that have attempted *failed connections* to more than  $k$  addresses in  $A$  during the day, where  $k$  is a system parameter defined by a pre-configured policy. Further connections initiated from the addresses in  $\Omega$  will be blocked by the gateway.

Some external hosts may temporarily behave abnormally, resembling a scan source. In order to release these addresses that are mistakenly placed into the scan list, each address in the list is associated with a timer, which is reset to  $T$  at the

beginning of each day. The address is removed from the list after timeout. Every time an address is re-inserted into the list, its timer is doubled. Therefore, normal users with temporary abnormal behavior will be removed from the blocking list, while persistent scanning sources will be punished.

For scan sources in  $\Omega$ , some of their connection attempts will be directed to a honeypot, which completes the connections and records the session data for analysis. An increasing number of scan sources sharing a similar traffic signature is an indication of worm activity. WEW generates a worm alert when the number of similar scan sources reaches a threshold  $n_0$ . We want to stress that, although widespread scan activity is not necessarily due to worm attack, a large-scale coordinated scan effort warrants a checkout. The progressive increase in the scan sources points towards a *likely* worm attack.

#### E. Sparsely Populated Address Space

WEW works better in a densely populated address space. If most of  $A_x$  is empty, then ICMP host-unreachable packets, instead of TCP RESET, are replied for most failed connection attempts. ICMP host-unreachable packets do not carry the information that is needed to create SYN/ACK<sub>1</sub>. To solve this problem, the gateway must be configured with a list of unused address prefixes, which is matched against the incoming SYN packets. If there is a match, the SYN is dropped and SYN/ACK<sub>1</sub> is created.

### III. ANALYSIS

#### A. Basics

The worm propagation can be roughly characterized by the classical simple epidemic model [1], [4], [7], [8].

$$\frac{di(t)}{dt} = r \frac{V}{N} i(t)(1 - i(t)) \quad (1)$$

where  $i(t)$  is the percentage of vulnerable hosts that are infected with respect to time  $t$ ,  $r$  is the rate at which an infected host scans the address space,  $N$  is the size of the address space, and  $V$  is the total number of vulnerable hosts. Suppose the worm starts with one infected host at  $t = 0$ . Solving (1), the number of hosts that are infected at time  $t$  is

$$n(t) = Vi(t) = V \frac{e^{r \frac{V}{N}(t-c)}}{1 + e^{r \frac{V}{N}(t-c)}} \quad (2)$$

where  $c = -\frac{N}{r \cdot V} \ln \frac{1}{V-1}$ . The time it takes for  $n$  vulnerable hosts to be infected is

$$t(n) = \frac{N}{r \cdot V} \ln \frac{n}{V-n} + c \quad (3)$$

#### B. System Sensitivity

Recall that WEW issues a worm alert when it detects  $n_0$  similar scan sources, where  $n_0$  is a system parameter. A performance metric, called *system sensitivity*, is defined as  $\frac{n_w}{n_0}$ , where  $n_w$  is the actual number of infected hosts at the time when the alert is issued. It is likely that  $n_w \neq n_0$  because there may be some infected hosts that have not yet been detected by WEW at the time of alert. The new metric indicates how

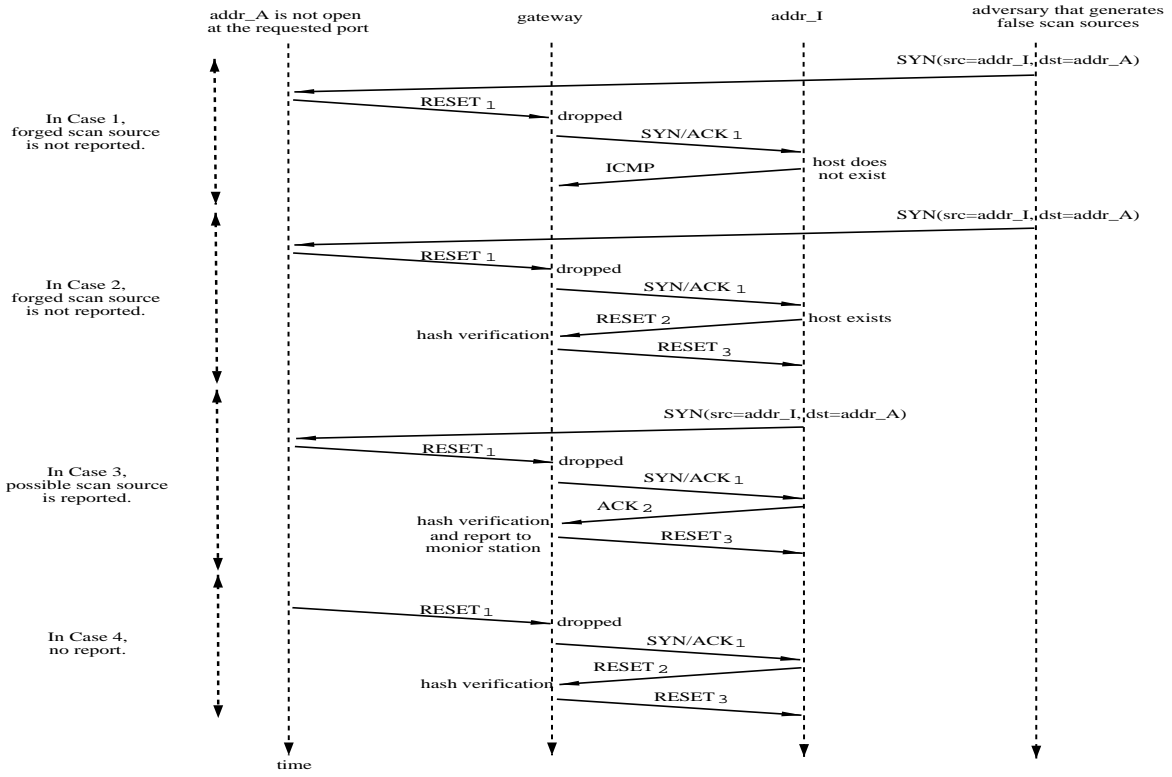


Fig. 2. Anti False Scan Sources

fast the system responds when the worm-like activities cross the “red” line and measures the gap between the number of actual infected hosts and the targeted goal ( $n_0$ ). Ideally, we want the system sensitivity to be close to one.

Suppose  $A$  is a small fraction of the IPv4 address space. Blocking scan sources by the gateway of WEW has negligible global impact on the worm propagation. The time it takes for the worm to infect  $n_0$  hosts is therefore

$$t(n_0) = \frac{N}{r \cdot V} \ln \frac{n_0}{V - n_0} + c$$

Among the  $n_0$  hosts, let  $x$  be the last to be infected. Let  $\Delta t$  be the time between  $x$  being infected and  $x$  being inserted by WEW into the scan list.  $x$  attempts  $r\Delta t$  random connections during  $\Delta t$ . Among those connections,  $r\Delta t \frac{A}{N}$  are made to  $A$ . Let  $\alpha$  be the percentage of connections that fail and are counted by the gateway of  $A$ . As we discussed previously,  $\alpha$  is likely to be large if  $A$  is densely populated or the gateway catches SYNs that are sent to unreachable addresses. The gateway inserts  $x$  into the scan list if  $\alpha r \Delta t \frac{A}{N} = k$ . We have

$$\Delta t = \frac{kN}{\alpha r A}$$

$x$  is detected by WEW at time  $t(n_0) + \Delta t$ . The  $n_0 - 1$  hosts that are infected before  $x$  must be detected before this time. Therefore, the *response time* for WEW to detect  $n_0$  similar scan sources and issue an alert for an on-going worm is about

$$t(n_0) + \Delta t.$$

$$\begin{aligned} t(n_0) + \Delta t &= \frac{N}{r \cdot V} \ln \frac{n_0}{V - n_0} - \frac{N}{r \cdot V} \ln \frac{1}{V - 1} + \frac{kN}{\alpha r A} \\ &= \frac{N}{r \cdot V} \ln \frac{n_0(V - 1)}{V - n_0} + \frac{kN}{\alpha r A} \end{aligned}$$

At the time when the alert is issued, the number of infected hosts is

$$\begin{aligned} n_w &= n(t(n_0) + \Delta t) = V \frac{e^{r \frac{V}{N}(t(n_0) + \Delta t - c)}}{1 + e^{r \frac{V}{N}(t(n_0) + \Delta t - c)}} \\ &= V \frac{n_0 e^{\frac{kV}{\alpha A}}}{V - n_0 + n_0 e^{\frac{kV}{\alpha A}}} \leq n_0 e^{\frac{kV}{\alpha A}} \end{aligned} \quad (4)$$

Therefore, we have an upper bound on system sensitivity.

$$\frac{n_w}{n_0} \leq e^{\frac{kV}{\alpha A}}$$

A nice property is that the system sensitivity is independent of the scan rate of a worm. A sufficient condition to achieve a target sensitivity of  $\gamma$  is

$$A \leq \frac{kV}{\alpha \ln \gamma}$$

For example, suppose  $V = 10^6$ ,  $k = 10$ ,  $\alpha = 0.6$ , and  $n_0 = 300$ . If the size of  $A$  is  $1.52 \times 10^7 < 2^{24}$ , WEW achieves a system sensitivity of at least 3.

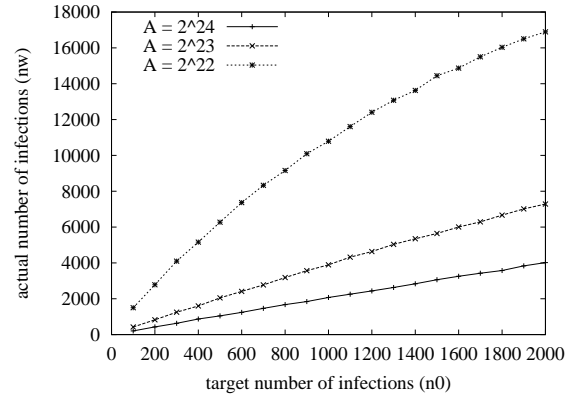
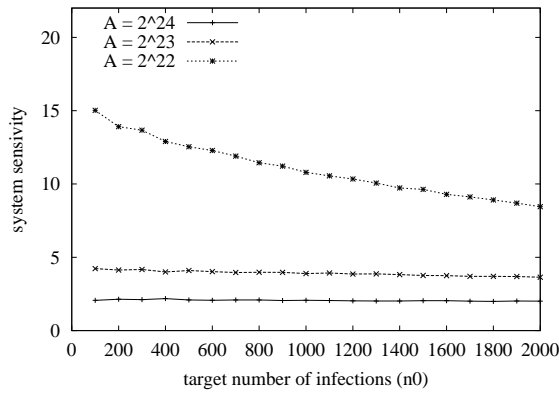


Fig. 3. system sensitivity

#### IV. SIMULATION RESULTS

This section presents the preliminary simulation results. The default simulation parameters are  $V = 10^6$ ,  $N = 2^{32}$ ,  $A = 2^{24}$ ,  $k = 10$ , and  $\alpha = 0.8$ . The default parameters are always assumed unless the figures indicate otherwise.

Fig. 3 presents the system sensitivity with respect to  $n_0$  and  $A$ . The left plot shows that the system sensitivity changes little with respect to  $n_0$ . For example, with  $A = 2^{24}$ , the sensitivity is around 2.1, decreasing only slightly as  $n_0$  increases. On the other hand, it increases significantly when  $A$  decreases. Hence, WEW works better when it monitors a larger address space. The right plot shows that  $n_0$  affects  $n_w$  almost linearly. Even if  $A$  is relatively small and thus the sensitivity value is big, as long as  $n_0$  is small, then  $n_w$  will still be small and account for only a small percentage of  $V$ . For example, if  $A = 2^{22}$  and  $n_0 = 300$ ,  $n_w$  accounts for just 0.4 percent of  $V$ .

Fig. 4 shows the time it takes WEW to report an on-going worm attack, with respect to the worm scanning rate  $r$  and  $n_0$ . For comparison, we also plot  $t(500,000)$ , the time it takes the worm to infect half of all vulnerable hosts. The propagation of the code-red worm [9] roughly corresponds to the data point at  $r = 65/\text{min}$ . The figure shows that, when  $n_0 = 300$ , it takes less than half of  $t(500,000)$  to issue an alert. The code red took about 9 hours to infect 250,000 hosts. The difference between  $t(250,000)$  and  $t(500,000)$  is rather small. Hence, based on the figure, WEW would have given a warning at the 4th hour when only less than 1000 hosts are infected, and it would have also given a list that contains many of those infected hosts. When an automated (or semi-automated) defense system is introduced in the future, WEW can be a valuable component that activates the defense measures and provides information about the attack traffic and the hosts that need to be disinfected.

#### V. CONCLUSION

This paper proposes an Internet-worm early warning system (WEW), designed as a component for a future worm defense system. It issues a warning about an on-going worm attack before it is fully propagated across the Internet.

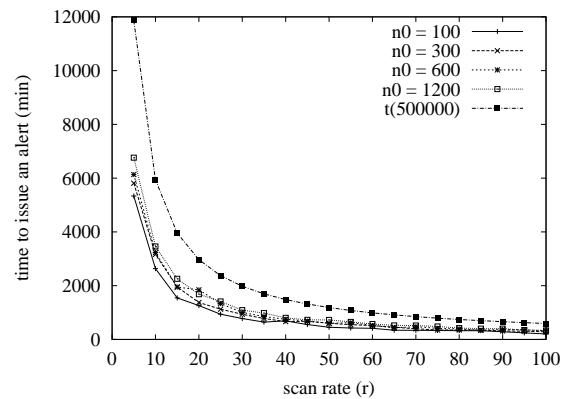


Fig. 4. system response time

#### REFERENCES

- [1] S. Staniford, V. Paxson, and N. Weaver, "How to Own the Internet in Your Spare Time," *Proc. of 11th USENIX Security Symposium, San Francisco*, August 2002.
- [2] M. Liljenstam, Y. Yuan, B. Premore, and D. Nicol, "A Mixed Abstraction Level Simulation Model of Large-Scale Internet Worm Infestations," *Proc. of 10th IEEE/ACM Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, October 2002.
- [3] C. C. Zou, W. Gong, and D. Towsley, "Code Red Worm Propagation Modeling and Analysis," *Proc. of 9th ACM Conference on Computer and Communication Security*, November 2002.
- [4] D. Moore, C. Shannon, G. M. Voelker, and S. Savage, "Internet Quarantine: Requirements for Containing Self-Propagating Code," *Proc. of IEEE INFOCOM'2003*, March 2003.
- [5] M. M. Williamson, "Throttling Viruses: Restricting Propagation to Defeat Malicious Mobile Code," *Proc. of Annual Computer Security Application Conference (ACSAC'02)*, December 2002.
- [6] Z. Chen, L. Gao, and K. Kwiat, "Modeling the Spread of Active Worms," *IEEE INFOCOM'03*, March 2003.
- [7] H. W. Hethcote, "The Mathematics of Infectious Diseases," *SIAM Review*, vol. 42, no. 4, pp. 599–653, 2000.
- [8] S. Chen and Y. Tang, "Slowing Down Internet Worms," *IEEE ICDCS'04*, March 2004.
- [9] C. E. R. Team, "CERT Advisory CA-2001-23 "Code Red" Worm Exploiting Buffer Overflow In IIS Indexing Service DLL," <http://www.cert.org/advisories/CA-2001-23.html>, July 2001.
- [10] D. E. Knuth, J. H. M. Jr., and V. R. Pratt, "Fast Pattern Matching in Strings," *SIAM Journal on Computing*, June 1977.