DAW: A Distributed Anti-Worm System

Shigang Chen Yong Tang
Department of Computer & Information Science & Engineering
University of Florida, Gainesville, FL 32611, USA
352 392 2713 (phone), 352 392 1220 (fax), {sgchen, yt1}@cise.ufl.edu

Abstract

A worm automatically replicates itself across the networks and may infect millions of servers in a short period of time. It is conceivable that the cyber-terrorists may use a wide-spread worm to cause major disruption to the Internet economy. Much recent research concentrates on propagation models and early warning, but the defense against worms is largely an open problem. We propose a distributed anti-worm architecture (DAW) that automatically slows down or even halts the worm propagation within an ISP (Internet Service Provider) network. New defense techniques are developed based on the behavioral difference between normal hosts and worm-infected hosts. Particularly, a worm-infected host has a much higher connection-failure rate when it randomly scans the Internet. This property allows DAW to set the worms apart from the normal hosts. We propose a temporal rate-limit algorithm and a spatial rate-limit algorithm, which makes the speed of worm propagation configurable by the parameters of the defense system. The effectiveness of the new techniques is evaluated analytically and by simulations.

Keywords

Internet worms, network security, rate-limit algorithms

I. Introduction

Ever since the Morris worm showed the Internet community for the first time in 1988 that a worm could bring the Internet down in hours [1], new worm outbreaks have occurred periodically. Take a few examples. On July 19, 2001, the code-red worm (version 2) infected more than 250,000 hosts in just 9 hours [2], [3]. Soon after, the Nimda worm raged on the Internet [4]. On January 25, 2003, a worm called SQLSlammer [5] caused widespread network congestion across Asia, Europe and the Americas. Santy worm, W32/Zafi.D, variants of W32/Sober, variants of W32/MyDoom, variants of W32/Bagle, and W32/Sasser were reported last year. A new MySQL UDF worm was reported early this year. Worms have beaten out viruses to become the top infectors of the Internet. A single worm is capable of automatically infecting millions of hosts in a short period of time, causing enormous damage [6]. It can steal sensitive information, remove files, slow down the network, or use the infected hosts to launch other attacks.

The most common way for a worm to propagate is to exploit a security loophole in certain version(s) of a service software to take control of the machine and copy itself over. For example, the Morris worm exploited a bug in *finger* and a trap door in *sendmail* of BSD 4.2/4.3. It also propagated through .rhosts/hosts.equiv and password guessing. The code-red worm took advantage of a buffer-overflow problem [7], [8] in the index server of IIS 4.0/5.0. Typically a worm-infected host scans the Internet for vulnerable systems. It chooses an IP address, attempts a connection to a service port (e.g., TCP port 80 in the case of *code red*), and if successful, carries out the attack. The above process repeats with different random addresses. As more and more machines are compromised, more and more copies of the worm are working together to reproduce themselves. An explosive epidemic is developed across the Internet.

There are few answers to the worm threat. One solution is to patch the software and eliminate the security defects [2], [4], [5]. That did not work because (1) software bugs seem always increase as computer systems become more and more complicated, and (2) not all people have the habit of keeping an eye on the patch releases. The patch for the security hole that led to the SQLSlammer worm was released half a year before the worm appeared, and still tens of thousands of computers were infected. Intrusion detection systems and anti-virus software may be upgraded to detect and remove a known worm, routers and firewalls may be configured to block the packets whose content contains worm signatures, but those happen after a worm has spread and been analyzed.

Much recent research on Internet worms concentrates on propagation modeling [6], [9], [10], [11], [12], [13] and early warning [11], [14], [15], [16]. The defense against worms is still an open problem. Moore et al. 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 [12]. Park et al. investigated worm containment methods in power-law Internet topologies and with partial deployment [17], [18]. Williamson proposed to modify the network stack such that the rate of connection requests to distinct destinations is bounded [19], [20]. It restricts a normal host in the same way it restricts a worm-infected host. Moreover, the approach becomes effective only after the majority of all Internet hosts is upgraded with the new network stack. The LaBrea approach [21] has a similar problem and can be easily circumvented by a worm that employs an early timeout mechanism. Staniford studied the containment of random scanning worms on a large enterprise network [22]. The model assumes the existence of a containment method that can block out an infected host after it scans around 10 addresses. However, such a method (without collateral damage of blocking normal hosts) is not given in the paper. Schechter et al.

[23] proposed a credit-based algorithm to limit the scan rate of a host, whose credit (i.e., allowance of making connections) is increased by one for each successful connection made and decreased by one for each failed connection made. This algorithm can be circumvented by an infected host that scans while making successful connections at the same rate. The signature-based defense systems require the worm samples to be captured before the attack signature can be generated [24], [25], [26], [27].

In this paper, we propose a distributed anti-worm architecture (DAW), which is designed for an Internet service provider (ISP) to provide anti-worm service to its customers. (From an ISP's point of view, the neighbor ISPs are also customers.) DAW is deployed at the ISP edge routers, which are under the same administrative control. It incorporates a number of new techniques that monitor the scanning activity within the ISP network, identify the potential worm threats, restrict the speed of worm propagation, and even halt the worms by blocking out scanning sources.

The proposed defense system separates the worm-infected hosts from the normal hosts based on their behavioral differences. Particularly, a worm-infected host has a much higher connection-failure rate when it randomly scans the Internet, whereas a normal user deals mostly with valid addresses due to the use of DNS (Domain Name System). This and other properties allow us to design the entire defense architecture based on the inspection of failed connection requests, which not only reduces the system overhead but also minimizes the disturbance to normal users. Combining a temporal rate-limit algorithm and a spatial rate-limit algorithm, DAW is able to tightly restrict the worm's scanning activity, while allowing the normal hosts to make successful connections at any rate. One important contribution of DAW is to make the speed of worm propagation configurable, no longer by the parameters of worms but by the parameters of DAW. While the actual values of the parameters should be set based on the ISP traffic statistics, we analyze the impact of these parameters on the performance of DAW and use simulations to study their suitable value ranges. The parameter settings used in this paper to evaluate the proposed algorithms are chosen based on the experimental data from real networks.

The rest of the paper is organized as follows. Section II describes the worm propagation model. Section III analyzes the differences between normal hosts and worm-infected hosts. Section IV presents the proposed distributed anti-worm architecture. Section V studies additional issues associated with DAW. Section VI addresses the inter-ISP worm infection. Section VII presents the simulation results. Section VIII draws the conclusion.

II. Modeling Worm Propagation

Based on the mathematical theory of infectious diseases [28], [29], [30], Kephart and White proposed a classic epidemiological model of computer viruses [31]. This model was later used to analyze the propagation behavior of Code-Red-like worms by Staniford et al. [6] and Moore et al. [32]. It can be written as

$$\frac{di(t)}{d(t)} = \beta i(t)(1 - i(t)) \tag{1}$$

where $i(t)$ is the fraction of vulnerable hosts that are infected with respect to time t, and β is the rate at which a worm-infected host detects other vulnerable hosts.

First we formally deduce the value of β. Some notations are defined as follows. r is the rate at which an infected host scans the address space. N is the size of the address space. V is the total number of vulnerable hosts.

At time t, the number of infected hosts is $i(t) \cdot V$, and the number of vulnerable but uninfected hosts is $(1 - i(t))V$. The probability for one scan message to hit an uninfected vulnerable host is $p = (1 - i(t))V/N$. For an infinitely small period dt, $i(t)$ changes by $di(t)$. During that time, there are $n = r \cdot i(t) \cdot V \cdot dt$ scan messages and the number of newly infected hosts is $n \times p = r \cdot i(t) \cdot V \cdot dt \cdot (1 - i(t))V/N = r \cdot i(t) \cdot (1 - i(t)) \frac{V^2}{N} dt$.[1] Therefore,

$$\begin{aligned} V \cdot di(t) &= r \cdot i(t) \cdot (1 - i(t)) \frac{V^2}{N} dt \\ \frac{di(t)}{dt} &= r \frac{V}{N} i(t)(1 - i(t)) \end{aligned} \tag{2}$$

The above equation agrees perfectly with our simulations. Solving the equation, we have

$$i(t) = \frac{e^{r\frac{V}{N}(t - t_0)}}{1 + e^{r\frac{V}{N}(t - t_0)}}$$

Let the number of initially infected hosts be v. $i(0) = v/V$, and we have $t_0 = -\frac{N}{r \cdot V} \ln \frac{v}{V - v}$. The time it takes for a percentage α ($\geq v/V$) of all vulnerable hosts to be infected is

$$t(\alpha) = \frac{N}{r \cdot V}(\ln \frac{\alpha}{1 - \alpha} - \ln \frac{v}{V - v}) \tag{3}$$

Suppose the worm attack starts from one infected host. $v = 1$. We have

$$t(\alpha) = \frac{N}{r \cdot V} \ln \frac{\alpha(V - 1)}{1 - \alpha} \tag{4}$$

[1] When $dt \to 0$, the probability of multiple scan messages hitting the same host becomes negligible.

The time predicted by Eq. (4) can be achieved only under ideal conditions. In reality, worms propagate slower due to a number of reasons. First, once a large number of hosts are infected, the aggressive scanning activities often cause widespread network congestions and consequently many scan messages are dropped. Second, when a worm outbreak is announced, many system administrators shut down vulnerable servers or remove the infected hosts from the Internet. Third, some types of worms enter dormant state after being active for a period of time. Due to the above reasons, the code red spread much slower than the calculation based on Eq. (4). A more sophisticated model that considers the first two factors can be found in [10], which fits better with the observed code-red data. An analytical active worm propagation model (AAWP) based on discrete times was proposed in [11], which addressed the localized scanning strategy.

Practically it is important to slow down the worm propagation in order to give the Internet community enough time to react when a new worm emerges. Eq. (4) points out two possible approaches: decreasing r causes $t(\alpha)$ to increase inverse-proportionally; increasing N causes $t(\alpha)$ to increase proportionally. In this paper, we use the first approach to slow down the worms, while relying on a different technique to halt the propagation. The idea is to block out the infected hosts and make sure that the scanning activity of an infected host does not last for more than a period of ΔT. Under such a constraint, the propagation model becomes

$$\frac{di(t)}{dt} = r\frac{V}{N}(i(t) - i(t - \Delta T))(1 - i(t)) \tag{5}$$

The above equation can be derived by following the same procedure that derives Eq. (2), except that at time t the number of infected hosts is $(i(t) - i(t - \Delta T)) \cdot V$ instead of $i(t) \cdot V$.

Theorem 1: If $\Delta T < (1 - \frac{v}{\alpha V})\frac{N}{rV}$, the worm will be stopped before a percentage α of all vulnerable hosts are infected.

Proof: Each infected host sends $r\Delta T$ scan messages, and causes $r\Delta T\frac{V}{N}$ (or less due to duplicate hits) new infections. For the worm to stop, we need $r\Delta T\frac{V}{N} < 1$. The total infections before the worm stops is no more than $\sum_{i=0}^{\infty} v(r\Delta T\frac{V}{N})^i = \frac{v}{1-r\Delta T\frac{V}{N}}$. If $\Delta T < (1 - \frac{v}{\alpha V})\frac{N}{rV}$, we have $\frac{v}{1-r\Delta T\frac{V}{N}} < \alpha V$. Namely, the worm stops before a percentage α of the vulnerable hosts are infected. □

III. Failure Rate

We present a new approach that measures the scanning activities by monitoring the failed connection requests, excluding those due to network congestion. Our discussion focuses on the worms that spread via TCP, which accounts for the majority of Internet traffic. However, the techniques can be easily applied to some

	avg. daily failure rate per host	worst daily failure rate per host
Net 1 (five Class C nets)	2.54	52
Net 2 (one Class C net)	7.72	77
Net 3 (two Class C nets)	2.79	63
	avg. daily failure rate of the whole network	worst daily failure rate of the whole network
Net 1 (five Class C nets)	670	880
Net 2 (one Class C net)	86	135
Net 3 (two Class C nets)	95	162

TABLE I

EXPERIMENTAL RESULTS: DAILY FAILURE RATES OF NORMAL HOSTS. THE DAILY FAILURE RATE OF A HOST IS THE NUMBER OF FAILED CONNECTIONS MADE BY THE HOST DURING A DAY.

UDP-based worms as well. We do not claim to handle all worms. Examples of what we do not consider are email worms and hit-list worms.

When a source host makes a connection request, a SYN packet is sent to a destination address. The connection fails if the destination host does not exist or does not listen on the port that the request is sent to. In the former case, an ICMP host-unreachable packet is returned to the source host; in the latter case, a RESET packet is returned.[2] These packets are called *failure replies*. The rate of failed connections made by a host is called the *failure rate*, which can be measured by monitoring the failure replies that are sent back to the host.

The failure rate of a normal host is likely to be low. For most Internet applications (www, telnet, ftp, etc.), a user types a machine name instead of a raw IP address to identify a server. The machine name is resolved by Domain Name System (DNS) for the IP address. If DNS can not find the address of a given name, the application will not make the connection. Hence, mistyping or stale web links do not result in failed connections. Moreover, a typical user has a list of favorite sites (servers) to which most connections are made. Since those sites are known to work most of the time, the failure rate for such a user will be low. If a connection fails due to network congestion, it does not affect the measurement of the failure rate because no ICMP host-unreachable or RESET packet is returned. We monitored three departmental networks on campus for a week. Table I shows our measured daily failure rates, which are very small. Failed connections made from a host to the same address in the same day is counted only once.

On the other hand, the failure rate of a worm-infected host is likely to be high. Most connections made

[2]For a UDP service, an ICMP port unreachable packet is returned if the host is not listening on the UDP port.

by a worm will fail if the destination addresses are randomly picked. Consider the infamous code-red worm, which uses 99 parallel threads to scan the Internet. We emulated its random scan and found that 99.6% of all connections made to random addresses on TCP port 80 fails. That is, the failure rate is 99.6% of the scanning rate. For worms targeting at software less popular than web servers, this figure will be even higher. The relation between the scanning rate r and the failure rate r_f of a worm is

$$r_f = (1 - \frac{V'}{N})r$$

where V' is the number of hosts that listen on the attacked port(s).[3] If $V' << N$, we have

$$r_f \approx r \tag{6}$$

Hence, measuring the failure rate of a worm gives a good idea about its scanning rate. Given the worm's aggressive scanning behavior, its failure rate is likely to be high, which sets it apart from the normal hosts. More importantly, an approach that restricts the failure rate will restrict the scanning rate, which slows down the worm propagation.

A worm may be deliberately designed to have a slow propagation rate in order to evade the detection, which will be addressed in Section IV-H.

IV. A Distributed Anti-Worm Architecture

A. Objectives

This section presents a distributed anti-worm architecture (DAW). Below are our main objectives.

- Slowing down the worm propagation to allow human reaction time. It took the code red just a few hours to achieve wide infection. More recent worms spread much faster. Our goal is to prolong that time to tens of days or even stop the worm propagation.
- Detecting potential worm activities and identifying likely offending hosts, which provides the security management team with valuable information in analyzing and countering the worm threat.
- Minimizing the performance impact on normal hosts and routers. Particularly, a normal host should be able to make successful connections at any rate, and the processing and storage requirements on a router should be minimized.

[3] $V \leq V'$ because not every host listens on the attacked port(s) is vulnerable.

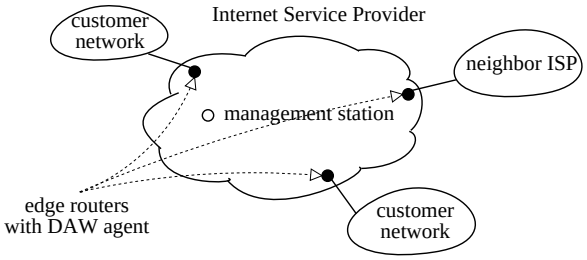

Fig. 1. Distributed Anti-Worm Architecture

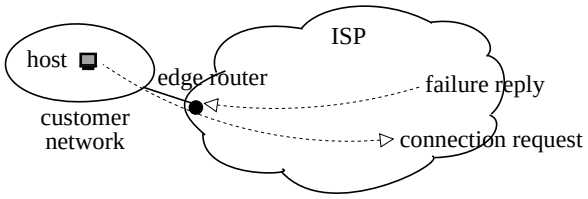

Fig. 2. An edge router monitors the failure replies for the customer it connects to.

B. DAW Overview

Most businesses, institutions, and homes access the Internet via Internet service providers (ISPs). An ISP network interconnects its customer networks, and routes the IP traffic between them. The purpose of DAW is to provide an ISP-based anti-worm service that slows or even stops Internet worms from spreading among the customer networks. DAW is practically feasible because its implementation is within a single administrative domain. It also has strong business merit since a large ISP has sufficient incentive to deploy such a system in order to gain marketing edge against its competitors.

As illustrated in Figure 1, DAW consists of two software components: a DAW agent that is deployed on all edge routers of the ISP and a management station that collects data from the agents. Each agent monitors the connection-failure replies sent to the customer network that the edge router connects to. It identifies the offending hosts in the customer network and measures their failure rates. If the failure rate of a host exceeds a pre-configured threshold, the agent randomly drops a minimum number of connection requests from that host in order to keep its failure rate under the threshold. A temporal rate-limit algorithm and a spatial rate-limit algorithm are used to constrain any worm activity to a low level over the long term, while accommodating the temporary aggressive behavior of normal hosts. Each agent periodically reports the observed scanning activity and the potential offenders to the management station. A continuous, steady increase in the gross scanning activity raises the flag of a possible worm attack. The worm propagation is further slowed or even stopped by blocking the hosts whose failure rates are persistently high.

Each edge router reads a configuration file from the management station about which addresses S and which ports P that it should monitor. S consists of all or some addresses belonging to the customer network. It provides a means to exempt certain addresses from DAW for research or other purposes. P consists of the port numbers to be protected such as 80/8080 for www and 23 for telnet. It should exclude the applications that are not suitable for DAW. An example is a hypothetical application runs with an extremely high failure rate, making

λ	defining the maximum failure rate allowed for an address in S
Ω	controlling the maximum number of failed connection requests allowed for an address per day
Φ	controlling the total number of failed connection requests allowed for a network per day
n	defining the number of days before the infected hosts are blocked

TABLE II
DAW PARAMETERS

normal hosts undistinguishable from worms targeting at the application. While DAW is not designed for all applications, it is very effective in protecting the network services whose clients require human interactions such as web browsing, which makes a greater distinction between normal hosts and worm-infected hosts.

Throughout the paper, when we say "a router receives a connection request", we refer to a connection request that enters the ISP from a customer network, with a source address in S and a destination port in P. When we say "a router receives a failure reply", we refer to a failure reply that leaves the ISP to a customer network, with a destination address in S and a source port in P (if it is a TCP RESET packet).

This paper does not address the worm activity within a customer network. A worm-infected host is not restricted in any way to infect other vulnerable hosts of the same customer network. DAW works only against the inter-network infections. The scanning rate of an infected host s is defined as the number of connection requests sent by s per unit of time to addresses outside of the customer network where s resides.

If a customer network has multiple edge routers with the same ISP, the DAW agent should be installed on all edge routers. If a customer network has connections with other ISPs that do not implement DAW, the network can be infected via those ISPs but is then restricted in spreading the worm to other customers of the ISP that does implement DAW. For the purpose of simplicity, we do not consider multi-homed networks in the analysis.

We discuss the details of DAW below. Some system parameters are listed in Table II for quick reference.

C. Measuring Failure Rate

Each edge router measures the failure rates for the addresses belonging to the customer network that the router connects to.

A failure-rate record consists of an *address* field s, a *failure rate* field f, a *timestamp* field t, and a *failure counter* field c. The initial values of f and c are zeros; the initial value of t is the system clock when the record is created. Whenever the router receives a failure reply for s, it calls the following function, which updates f

each time c is increased by 100. β is a parameter between 0 and 1.

Update_Failure_Rate_Record( )

(1) $c \leftarrow c + 1$

(2) **if** (c is a multiple of 100)

(3) $f' \leftarrow 100/(\text{the current system clock} - t)$

(4) **if** ($c = 100$)

(5) $f \leftarrow f'$

(6) **else**

(7) $f \leftarrow \beta \times f + (1 - \beta) \times f'$

(8) $t \leftarrow$ the current system clock

It is unnecessary to create individual failure-rate records for those hosts that occasionally make a few failed connections. An edge router maintains a hash table H. Each table entry is a failure-rate record without the address field. When the router receives a failure reply, if the destination address does not have its own failure record, the router hashes the address to a table entry in H and calls Update_Failure_Rate_Record() on that entry. Each entry therefore measures the combined failure rate of roughly $A/|H|$ addresses, where A is the size of the customer network and $|H|$ is the size of the hash table.

Only when the rate of a hash-table entry exceeds a threshold λ (e.g., one per second), the router creates failure-rate records for individual addresses of the entry. A failure-rate record is removed if its counter c registers too few failed connections in a period of time.

D. Basic Rate-Limit Algorithm

Let F_λ be the set of addresses whose failure rates are larger than λ. Every address in F_λ has an individual failure-rate record because the hash-table entry that the address maps to must have a rate exceeding λ. For each $s \in F_\lambda$, the router reduces its the failure rate below λ by rate-limiting the connection requests from s. A token bucket is used. Let $size$ be the bucket size, $tokens$ be the number of tokens, and $time$ be a timestamp whose initial value is the system clock when the algorithm starts.

Upon receipt of a failure reply to s

(1) $tokens \leftarrow tokens - 1$

Upon receipt of a connection request from s

(2) $\Delta t \leftarrow$ the current system clock $- \, time$

(3) $tokens \leftarrow \min\{tokens + \Delta t \times \lambda, \; size\}$

(4) $time \leftarrow$ the current system clock

(5) **if** ($tokens \geq 1$)

(6) forward the request

(7) **else**

(8) drop the request

We want to emphasize that the above algorithm with two subroutines is not a traditional token-bucket algorithm that buffers the traffic bursts and releases them at a fixed rate. The purpose of our algorithm is not to shape the flow of incoming failure replies but to restrict the "creation" of the failure replies. It ensures that the failure rate of any address in S stays below λ. This effectively restricts the scanning rate of any worm-infected host according to Eq. (6). Consequently, the speed of worm propagation is no longer determined by the worm parameters set by the attackers, but by the DAW parameters set by the ISP administrators. In the rest of the section, we will propose more advanced rate-limit algorithms to give the defenders greater control.

All rate-limit algorithms in the paper are performed on individual addresses. They are not performed on the failure-rate records in the hash table; otherwise, requests from many innocent hosts would have been blocked when one scan source was mapped to the same hash-table entry.

Our basic rate-limit algorithm restricts the failure rate but not the success rate; a host can make successful connections at any rate. In comparison, Williamson's approach bounds the rate of successful connections in the same way it bounds the rate of worm scan [19], [20].

Next, we explore the temporal behavior difference between normal hosts and worm-infected hosts to further tighten the worm scan rate by a temporal rate-limit algorithm.

E. Temporal Rate-Limit Algorithm

A normal user behaves differently from a worm that scans the Internet tirelessly, day and night. A user may generate a failure rate close to λ for a short period of time, but that can not last for every minute in 24 hours of a day. While we set λ large enough to accommodate temporary aggressiveness in normal behavior, the rate over a long period can be tightened appropriately, which does not affect a normal user but reduces the long-run average scan rate of a worm-infected host. Let Ω be the system parameter that controls the maximum number

of failed connection requests allowed for an address per day. Let D be the time of a day. Ω can be set much smaller than λD.

At the start of each day, the counters (c) of all failure-rate records and hash-table entries are reset to zeros. The value of c always equals the number of failed requests that have happened during the day. We now require that a hash-table entry creates failure-rate records for individual addresses when either $f > \lambda$ or $c > \Omega$.

A temporal rate-limit algorithm is designed to bound the maximum number of failed requests per day. Let F_Ω be the set of addresses that satisfy the following two conditions: $\forall s \in F_\Omega$, (1) s has an individual failure-rate record, and (2) either the failure rate of s is larger than λ or the counter of s reaches $\Omega/2$. It is obvious that $F_\lambda \subseteq F_\Omega$.

Upon receipt of a failure reply to s

(1) $tokens \leftarrow tokens - 1$

Upon receipt of a connection request from s

(2) $\Delta t \leftarrow$ the current system clock $- \, time$

(3) **if** ($c \leq \Omega/2$)

(4) $\quad tokens \leftarrow \min\{tokens + \Delta t \times \lambda,\ size\}$

(5) **else**

(6) $\quad \lambda' \leftarrow \frac{\Omega - c - \max\{tokens,\ 0\}}{\text{the end of the day} \ - \ time}$

(7) $\quad tokens \leftarrow \min\{tokens + \Delta t \times \lambda',\ size\}$

(8) $time \leftarrow$ the current system clock

(9) **if** ($tokens \geq 1$)

(10) $\quad$ forward the request

(11) **else**

(12) $\quad$ drop the request

The temporal rate-limit algorithm constrains both the maximum failure rate and the maximum number of failed requests per day. When it is used, the basic rate-limit algorithm is not necessary. Before c reaches $\Omega/2$, the failure rate can be as high as λ. After that, the algorithm spreads the remaining "quota" ($\Omega - c - \max\{tokens,\ 0\}$) on the rest of the day, which ensures that connections will be forwarded throughout the day. Line 6 ensures that the algorithm never blocks a host completely. Because normal hosts rarely make failed connections, the impact of the algorithm on normal hosts is expected to be insignificant. It should also be

pointed out that the algorithm places no restriction on the success rate.. *A normal host can make successful connections at any rate during the day (e.g., browsing the favorite web sites that are up) because the constraint is on failure replies only*.

Theorem 2: When the temporal rate-limit algorithm is used, the number of failure replies for any address does not exceed $2\Omega + rT$ in a day, where r is the rate at which the host makes connection requests and T is the round trip delay in the ISP.

The proof can be found in the appendix. rT is normally small because the typical round trip delay across the Internet is in tens or hundreds of milliseconds. Hence, if $\Omega = 100$, the average scanning rate of a worm is effectively limited to about $2\Omega/D = 0.14/min$. In comparison, Williamson's experiment showed that the scanning rate of the code red was at least $200/sec$ [19], which is more than 85,000 times faster. Yet, it took the code red hours to spread, suggesting the promising potential of using the temporal rate-limit algorithm to slow down worms.

Additional system parameters that specify the maximum numbers of failed requests in longer time scales (week or month) can further increase the worm propagation time.

In practice, all system parameters in DAW should be set based on the ISP's traffic measurement. They should be larger than the worst-case numbers of a normal host. For example, our 7-day measurement on the campus network shows that the maximum number of failed connections by a normal host during a day is only 77. For this network, $\Omega = 100$ is appropriate.

F. Recently Failed Address List

If a major web server such as Yahoo or CNN is down, an edge router may observe a significant surge in failure replies even though there is no worm activity. To solve this problem, each edge router maintains a recently failed address list (RFAL), which is emptied at the beginning of each day. When the router receives a failure reply from address d, it matches d against the addresses in RFAL. If d is in the list, the router skips all DAW-related processing. Otherwise, it inserts d into RFAL before processing the failure reply. If RFAL is full, d replaces the oldest entry in the list.

When a popular server is down, if it is frequently accessed by the hosts in the customer network, the server's address is likely to be in RFAL and the failure replies from the server will not be repetitively counted. Hence, the number of failed requests allowed for a normal host per day can be much larger than Ω. It effectively places no restriction on keeping trying a number of favorite sites that are temporarily down. On the other hand, given

the limited size of RFAL and the much larger space of IPv4 (2^{32}), the random addresses picked by worms have a negligibly small chance to fall in the list.

G. Spatial Rate-Limit Algorithm

Refer back to Table I. Consider Net1 whose size is $A = 5 \times 256 = 1,280$. Even though there was one normal host making 52 failed connections during a day, not every host did that. The daily number of failed connections from Net1 was at most 880. If we set $\Omega = 100$ to accommodate the aggressive behavior of some normal hosts, the combined daily failure rate of the whole network can be tightened far less than $A\Omega$ $(= 128,000)$ without adverse impact on other normal hosts.

The proposed temporal rate-limit algorithm regulates each individual infected host. DAW uses a spatial rate-limit algorithm to constrain the combined scanning rate of all infected hosts in a customer network. A failure reply will be first processed by the temporal algorithm and then by the spatial algorithm if it is activated.

Let Φ be a system parameter that controls the total number of failed requests allowed for a customer network per day. It may vary for different customer networks based on their sizes. Once the number of addresses inserted to RFAL exceeds Φ, the system starts to create failure-rate records for all addresses that receive failure replies, and activates the spatial algorithm. If there are too many records, it retains those with the largest counters. Let F_Φ be the set of addresses whose counters exceed a small threshold τ (e.g., 20), which excludes the obvious normal hosts. Let tc be the total number of failure replies sent to F_Φ since the spatial algorithm is activated. The spatial rate-limit algorithm is performed only on addresses in F_Φ. Its pseudo code is the same as that of the temporal algorithm except that s, Ω, and c are replaced by F_Φ, Φ, and tc, respectively. If there are a large number of infected hosts, causing the spatial algorithm to drop an excessive amount of requests, the router should temporarily block the addresses whose failure-rate records have the largest counters.

Theorem 3: When the spatial rate-limit algorithm is used, the total number of failure replies per day for all infected hosts in a customer network is bounded by $2\Phi + mr'T$, where m is the number of addresses in F_Φ, r' is the scanning rate of an infected host after the temporal rate-limit algorithm is applied, and T is the round trip delay of the ISP.

The proof is omitted, which is very similar to the proof of Theorem 2 in the appendix. $mr'T$ is likely to be small because both r' and T are small. Below we analyze how the value of Φ will affect the worm propagation based on a simplified model. A more general model will be used in the simulations.

Suppose there are k vulnerable customer networks, each with V/k vulnerable hosts. Once a host is infected,

we assume all other vulnerable hosts of the same customer are infected immediately because DAW does not restrict the scanning activity within the customer network. Based on Theorem 3, the combined scanning rate of all vulnerable hosts in a customer network is $(2\Phi + mr'T)/D \approx 2\Phi/D$. Let $j(t)$ be the fraction of customer networks that are infected by the worm with respect to time t.

At time t, the number of infected customer networks is $j(t) \cdot k$, and the number of uninfected networks is $(1 - j(t))k$. The probability for one scan message to hit an uninfected vulnerable host and thus infect the network where the host resides is $(1 - j(t))V/N$. For an infinitely small period dt, $j(t)$ changes by $dj(t)$. During that time, there are $\frac{2\Phi}{D} \cdot j(t) \cdot k \cdot dt$ scan messages and the number of newly infected networks is $\frac{2\Phi}{D} \cdot j(t) \cdot k \cdot dt \cdot (1 - j(t))V/N = \frac{2\Phi}{D} \cdot j(t) \cdot (1 - j(t))\frac{Vk}{N}dt$. [4] Therefore,

$$
\begin{aligned}
k \cdot dj(t) &= \frac{2\Phi}{D} \cdot j(t) \cdot (1 - j(t))\frac{Vk}{N}dt \\
\frac{dj(t)}{dt} &= \frac{2V\Phi}{ND} j(t)(1 - j(t)) \\
j(t) &= \frac{e^{\frac{2V\Phi}{ND}(t-t_0)}}{1 + e^{\frac{2V\Phi}{ND}(t-t_0)}}
\end{aligned}
$$

Assume there is one infection at time 0. We have $t_0 = -\frac{ND}{2V\Phi} \ln \frac{1}{k-1}$. The time it takes to infect α percent of all networks is

$$
t(\alpha) = \frac{ND}{2 \cdot V\Phi} \ln \frac{\alpha(k-1)}{1-\alpha}
$$

Suppose an ISP wants to ensure that the time for α percent of networks to be infected is at least γ days. The value of Φ should satisfy the following condition.

$$
\Phi \leq \frac{N}{2 \cdot V\gamma} \ln \frac{\alpha(k-1)}{1-\alpha}
$$

which is not related to how the worm behaves.

H. Blocking Persistent Scanning Sources

The edge routers are configured to block out the addresses whose counters (c) reach Ω for n consecutive days, where n is a system parameter. By Eq. (5) and Theorem 1, the worm propagation may be stopped if the infected hosts are blocked out after n days of activity.

The worm propagates slowly under the temporal rate-limit algorithm and the spatial rate-limit algorithm. It gives the administrators sufficient time to study the traffic of the hosts to be blocked, perform analysis to determine whether a worm infection has occurred, and decide whether to approve or disapprove the blocking.

[4] The probability of multiple external infections of the same network is negligible when $dt \to 0$.

Once the threat of a worm is confirmed, the edge routers may be instructed to reduce n, which increases the chance of fully stopping the worm.

Suppose a worm scans more than Ω addresses per day. The worm propagation can be completely stopped if each infected customer network makes less than one new infection on average before its infected hosts are blocked. The number of addresses scanned by the infected hosts from a single network during n days is about $2n\Phi$ by Theorem 3. Each message has a maximum probability of V/N to infect a new host. Hence, the condition to stop a worm is

$$2n\Phi\frac{V}{N} < 1$$

The expected total number of infected networks is bounded by

$$\sum_{i=0}^{\infty}(2n\Phi\frac{V}{N})^i = \frac{1}{1-2n\Phi\frac{V}{N}}$$

When $2n\Phi\frac{V}{N} \geq 1$, the worm may not be stopped by the above approach alone. However, the significance of blocking infected hosts should not be under-estimated as it makes the worm-propagation time longer and gives human or other automatic tools more reaction time.

If the scanning rate of a worm is below Ω per day, the infected hosts will not be blocked. DAW relies on a different approach to address this problem. During each day, an edge router e measures the total number of connection requests, denoted as $n_c(e)$, and the total number of failure replies, denoted as $n_f(e)$. Note that only the requests and replies that match S and P (Section IV-B) are measured. The router sends these numbers to the management station at the end of the day. The management station measures the following ratio

$$\frac{\sum_{e\in E} n_f(e)}{\sum_{e\in E} n_c(e)}$$

where E is the set of edge routers. If the ratio increases significantly for a number of days, it signals a potential worm threat. That is because the increase in failed requests steadily outpaces the increase in issued requests, which is possibly the result of more and more hosts being infected by worms.

The management station then instructs the edge routers to identify potential offenders whose counters (c) have the highest values. Additional potential offenders are found as follows. After a vulnerable server is infected via a port that it listens to, the server normally scans the Internet on the same port to infect other servers. Based on this observation, when an edge router receives a RESET packet (with source port p) to an address s in its customer network, it sends a SYN packet to check if s is also listening on port p. If it is, the router marks s as a potential offender and creates a failure-rate record, which measures the number of failed

connections from s. At the end of each day, the management station collects the potential offenders from all edge routers. Those with the largest counters are presented to the administrators for traffic analysis. The management station may instruct the edge routers to block them if the worm threat is confirmed.

Although a blocked host can not *issue* connection requests before it is unblocked, it can *accept* connection requests at any rate and its return packets of the accepted connections are not blocked. Namely, its role of a server is unchanged.

An alternative to complete blocking is to apply a different, small Ω value (e.g., 20) on those addresses, which restricts the worm-infected hosts more tightly while leaving certain room against false positives since the hosts can still make as many successful connections as they want, with occasional failures.

V. Additional Issues

A. Overhead

First, to implement DAW, an edge router needs to process connection requests and failure replies, which account for a tiny fraction of all traffic passing through the router (Table III). To identify the packet type, the router needs to check the next-protocol field in the IP header and then the type or flags in the next header, which is not expensive. Second, all DAW operations are very simple, which is evident from the pseudo code of algorithms. Third, the number of failure records kept for DAW is likely to be small. Failure-rate records are only maintained for a subset of addresses on the customer network. Records are removed if they register too few failed connections in a period of time after creation. Normal hosts rarely make failed connections. As an example, the average number of failed connections (made to different addresses) is less than 5 per host per day on the UF campus network. Therefore, most normal hosts will not have records. The number of failure-rate records grows with the number of worm-infected hosts, which must be monitored anyway. Finally, we performed a simulation on a low-end Dell-PC "edge router" (2.40GHz/512M), which is able to process 2.0×10^6 connection requests per second under DAW, when the number of failure-rate records is 1,000. Due to space limitation, we defer more detailed performance studies to the future work.

B. Forged Failure Replys

To prevent forged failure replies from being counted, one approach is to keep a table of recent connection requests from any source address in S to any destination port in P during the past 45 seconds (roughly the MRTT of TCP). S and P are defined in Section IV-B. Each table entry contains a source address, a source

port, a destination address, and a destination port, identifying a connection request. Only those failure replies that match the table entries are counted. An alternative approach is to extend the failure-rate record by adding two fields: one (x) counting the number of connection requests from s and the other (y) counting the number of successful connections, i.e., TCP SYN/ACK packets sent to s, where s is the address field of the record. An invariant is maintained such that the number of failed connections plus the number of successful connections does not exceed the number of connection requests, i.e., $c + y \leq x$. A failure reply is counted ($c \leftarrow c + 1$) only when the invariant is not violated.

C. Failure Reply Suppression

Many firewalls are configured to suppress failure replies. Particularly, because attacks routinely use ICMP as a reconnaissance tool, many organizations block outbound ICMP host-unreachable packets. While those ICMP packets should not be seen by their destinations, they are important to DAW. To solve this problem, for the customers that fully participate in DAW, when their firewalls suppress ICMP host-unreachable packets, they are required to send FailLog messages instead, which are counted and then suppressed by the edge routers before leaving the ISP.

D. Warhol Worm and Flash Worm

The Warhol worm and the Flash worm are hypothetical worms studied in [6], which describes a number of highly effective techniques that the future worms might use to infect the Internet in a very short period of time, leaving no room for human actions.

In order to improve the chance of infection during the initial phase, the Warhol worm first scans a pre-made list of (e.g., 10000 to 50000) potentially vulnerable hosts, which is called a *hit-list*. After that, the worm performs *permutation scanning*, which divides the address space to be scanned among the infected hosts. One way to generate a hit-list is to perform a scan of the Internet before the worm is released [6]. Consider the worm propagation in an ISP that deploys DAW. To generate a hit-list for the ISP, it will take about $N/2\Omega$ days. Suppose $\Omega = 100$ and $N = 2^{24}$. That would be 229.8 years. A distributed scan would require a very large number of cooperative hosts in order to cut this time down. Even if the hit-list can be generated by a different means, the permutation scanning is less effective under DAW. For instance, even after 1000 vulnerable hosts are infected, they can only probe about $1000 \times 2\Omega = 6 \times 10^5$ addresses a day if only the temporal rate-limit algorithm is considered. The number can be much smaller if the spatial algorithm is taken into consideration.

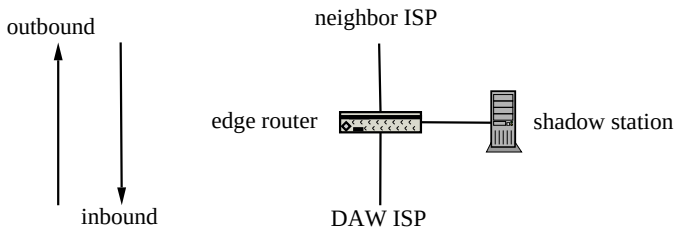

Fig. 3. shadow station

Suppose the size of the ISP is $2^{24} \approx 1.68 \times 10^7$, which is far larger than 6×10^5. Duplicate hits are not a serious problem, which means the gain by permutation scanning is small. Without DAW, it will be a different matter. If the scanning rate is $200/sec$, it takes less than 6 hours for 1000 infected hosts to make 2^{32} probes, and duplicate hits are very frequent.

The Flash worm assumes a hit-list L including most servers that listen on the targeted port. Hence, random scanning is completely avoided; the worm scans only the addresses in L. As more and more hosts are infected, L is recursively split among the newly infected hosts, which scan only the assigned addresses from L. The Flash worm may require a prescan of the entire address space before it is released. Such a prescan takes too long under DAW.

VI. Inter-ISP Infection

The techniques developed in Section IV can effectively slow down or even halt the worm propagation within the same ISP, but they are less effective against the infection from outside of the ISP. New techniques are developed in this section to solve the inter-ISP infection problem.

A. Performance Issue and Shadow Station

An ISP that deploys the DAW system is called a *DAW ISP*. Each neighbor ISP is treated as a special customer network. From the DAW ISP's point of view, its neighbor ISPs represent the rest of the Internet and therefore represent the much larger external address space, which may contain a very large population of worm-infected hosts, possibly in the number of hundreds of thousands. Figure 3 shows an edge router that connects to a neighbor ISP. We use "outbound" to mean the direction from the DAW ISP to the neighbor ISP and "inbound" the opposite direction. The edge router performs the DAW function by monitoring the outbound failure replies and rate-limit the inbound connection requests.

Based on the netflow records from the main gateway of the University of Florida, we found that the SYN packets (connection requests) account for only 4.86% of all packets and, in terms of bytes per second, they

percentage of packets that are SYN	4.86%
percentage of traffic volume (in bytes) that are SYN	0.277%

TABLE III

EXPERIMENTAL RESULTS: SYN TRAFFIC THROUGH THE MAIN GATEWAY OF UF

account for only 0.277% of the traffic volume, as shown in Table III. The number of failure replies is much smaller than the number of connection requests. Therefore, DAW only process a small fraction of the packets passing the edge router. Moreover, the temporal/spatial algorithms are very simple.

Nevertheless, we propose to offload the DAW function from the edge router to a shadow station, which is illustrated in Figure 3. More specifically, for each outbound failure reply whose source address belongs to the DAW ISP, the edge router forwards a copy to the shadow station; for each inbound connection request whose destination address belongs to the DAW ISP, the edge router redirects it to the shadow station, where all DAW-related operations are performed. The connection requests that pass the rate-limit algorithms will be sent back to the edge router and then routed into the DAW ISP.

Shadow stations may also be implemented for edge routers that connect to large enterprise networks.

B. Anti Inter-ISP Infection

The local customer networks of the DAW ISP have relatively small address spaces that can be watchdoged by the temporal and the spatial rate-limit algorithms. However, these algorithms are less effective when they are implemented on the edge routers connecting to the neighbor ISPs. Suppose there are a million infected hosts on the Internet and the DAW ISP is a Class A network, which has $2^{24} = 16,777,216$ addresses. The temporal algorithm allows each infected host to scan up to 2Ω addresses per day. If $\Omega = 100$, then 200 million addresses are scanned in a single day, which is far more than the total number of addresses in the DAW ISP. The spatial algorithm will not solve the problem, either. That is because, in order to avoid blocking normal hosts, the algorithm is not applied on failure records whose counters are below a threshold (e.g., 20). Hence, one million external infected hosts would be allowed to scan up to 20 million addresses per day, still more than the total number of addresses in the DAW ISP.

To deal with the inter-ISP infection problem, we develop additional defense mechanisms as follows. The shadow station maintains a bit map (called *failure bitmap*), where each bit represents an address of the DAW ISP. The size of the failure bitmap is 2 Megabytes for a class A network. All bits are reset to zero at the beginning of a day. When an outbound failure reply is received, the bit for the source address is set to one,

meaning that the address may have been scanned. By default, the shadow station runs the temporal/spatial algorithms in the normal mode. However, when a noticeable portion of the DAW ISP has been scanned and the percentage of the failure bitmap set to one reaches θ (which is a system parameter such as 0.1%), the shadow station enters the *aggressive mode*, as worm propagation may potentially be undergoing.

One way to implement the aggressive mode is to perform the spatial algorithm on all failure records instead of only those whose counters are above the threshold. This approach is likely to block some normal hosts that happen to make failed connections. A more sophisticated approach is based on the observation that all known worms do not kill the service software they infect for two reasons: (1) the infection is performed on a thread or child process of the server, which neither blocks nor kills the entire service; and (2) the stealth requirement of worm propagation makes it undesirable to disrupt the server's normal operations. Therefore the infected hosts are open on the port that they scan. This observation can be utilized to aim the spatial algorithm towards a narrow population of likely offenders. The algorithm for the aggressive mode is is described below.

Upon receipt of a connection request from an external address s

(1) **if** ($s \in F_\Phi$)
(2) perform the spatial rate-limit algorithm on the request
(3) **else**
(4) **if** (s is open at the destination port of the request)
(5) create a failure record for s with $c = 0$
(6) add s to F_Φ
(7) perform the spatial rate-limit algorithm on the request
(8) **else**
(9) forwards the request back to the edge router

The remaining problem is Line 4, how to determine if s is open at a specific port p. We use the cryptographic cookie approach. The shadow station sends a forged connection request to s at port p. The request carries a cookie, e.g., SYN cookie [33] stored in the sequence number field of the TCP header. The cookie is generated as a keyed hash of s, p, a secret, a timestamp, etc. Note that the shadow station does not store the half-opened connection because it is a forged connection request. Within a short timeout period (e.g., one second[5]), if a connection reply (e.g., SYN/ACK) comes back from s and the cookie (in the acknowledgement field) is

[5]The Internet round trip delay is typically in tens or hundreds of millisecond, according to our experiment of pinging random addresses.

correctly verified, then s is open at p; if a failure reply (e.g., RESET) comes back from s and the cookie is correctly verified, or no reply comes back, then s is not open at p.

The number of connection requests forged by the shadow station is bounded by the number of connection requests it receives. The latter accounts for a small portion of the overall traffic, as demonstrated in Table III. Consequently, the forged traffic volume is insignificant and this traffic is introduced only when the shadow station enters the aggressive mode.

VII. SIMULATION

We use simulations to evaluate the performance of DAW. We simulate an ISP that performs the algorithms of DAW to protect its customer networks. Each data point in the figures is the average of 2,000 simulation runs.

The simulation setup is described as follows. The size of the ISP address space is 2^{24}. The number of vulnerable servers targeted by a worm is 1 million on the whole Internet. The number of vulnerable customer networks in the ISP is $k = 2000$. The average number of vulnerable hosts in these networks is $z = 5$.[6] The numbers of vulnerable servers in different networks follow an exponential distribution, suggesting a scenario where most customer networks have five or less vulnerable servers, but some have large numbers of such servers. Suppose the worm uses a Nimda-like algorithm that aggressively searches the local-address space. We assume that once a vulnerable host of a customer network is infected, all vulnerable hosts of the same network are infected immediately. We also ignore the packet delay. These are conservative assumptions because they make the worm propagates faster. If DAW works well under these assumptions, it should work even better in practice. Unless specified otherwise, the default DAW parameters are $\lambda = 1$/sec, $\Omega = 100$, $\Phi = 1000$, and $n = 7$ days. They are chosen based on Table I. We will vary each parameter in a wide range to study the performance trend of DAW. The scan rate of an infected host is set at $10/sec$, which is in fact not an important parameter because the performance of DAW is insensitive to the worm scanning rate due to rate limiting.

Subsections VII-A through VII-E study the worm propagation within the ISP, starting from one infected customer network. We take an incremental approach by first investigating the effectiveness of the temporal or spatial rate-limit algorithm alone, then putting them together, and finally, applying the whole DAW. Subsection VII-F shows the misblocking ratio of a normal host. Subsection VII-G studies the inter-ISP worm propagation.

[6]The ISP has $\frac{1}{256}$ of the Internet address space. Discounting the unusable addresses (reserved, multicast, private, unallocated addresses), we let the ISP have $\frac{1}{100}$ of all vulnerable hosts.

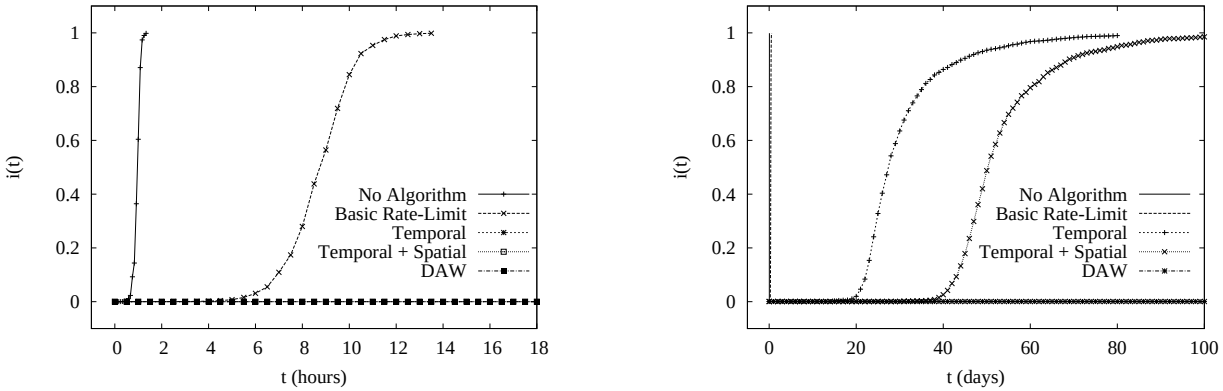

Fig. 4. worm-propagation comparison

A. Propagation-Time Comparison

Figure 4 compares the fraction $i(t)$ of vulnerable hosts that are infected over time t in five different cases: 1) no algorithm is used, 2) the basic rate-limit algorithm is implemented on the edge routers, 3) the temporal rate-limit algorithm is implemented, 4) both the temporal and spatial rate-limit algorithms are implemented, or 5) DAW (i.e., Temporal, Spatial, and blocking persistent scanning sources) is implemented. Note that all algorithms limit the failure rates, not the connection request rates, and the spatial rate-limit algorithm is applied only on the hosts whose failure counters exceed a threshold $\tau = 20$. Two graphs show the simulation results in different time scales. The lefthand graph is from 0 to 18 hours, and the righthand is from 0 to 100 days. The shape of the curve "No Algorithm" depends on the worm's scanning rate, which is 10/sec in our simulation. The other four curves are independent of the worm's scanning rate; they depend only on DAW's parameters, i.e., λ, Ω, Φ, and n. The figure shows that the basic rate-limit algorithm slows down the worm propagation from minutes to hours, while the temporal rate-limit algorithm slows down the propagation to tens of days. The spatial rate-limit algorithm makes further improvement on top of that — it takes the worm 41 days to infect 5% of the vulnerable hosts, leaving sufficient time for human intervention. Moreover, with persistent scanning sources being blocked after 7 days, DAW is able to stop the worm propagation at $i(t) = 0.005$.

B. Temporal Rate-Limit Algorithm Alone

Figure 5 demonstrates the performance of the temporal rate-limit algorithm with respect to the parameter Ω. The y axis is the time it takes the worm to infect 5% of vulnerable hosts (called *5% propagation time*).

As expected, the propagation time decreases when Ω increases. The figure also shows that the temporal algorithm alone already performs very well for modest-size ISPs. When $k = 2000$, $z = 5$ and $\Omega = 100$, the 5% propagation time is 24.5 days. Note that k is not the number of customer networks, but the number of

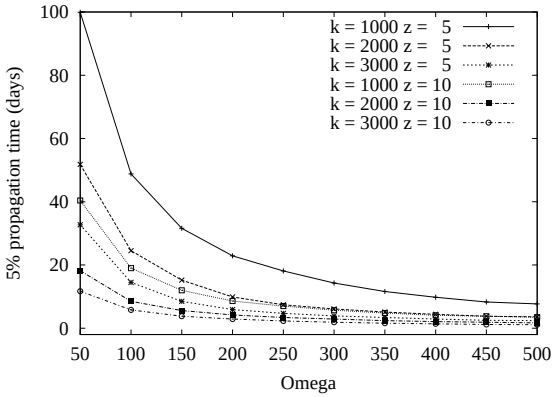

Fig. 5. effectiveness of the temporal rate-limit algorithm

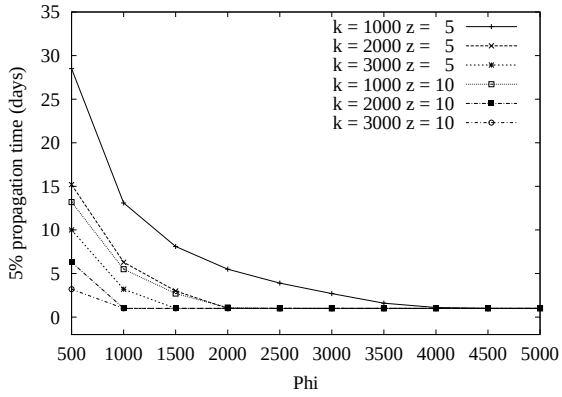

Fig. 6. effectiveness of the spatial rate-limit algorithm

customer networks that have vulnerable hosts. If the total number of vulnerable hosts ($k \times z$) in the ISP stays the same, the algorithm works better for a larger k.

While Ω determines the maximum long-term failure rate that is allowed for a host, the other parameter λ determines the maximum short-term failure rate. λ has little impact on the propagation time, which mostly depends on the worm's long-term scanning rate, not the short-term rate. The simulation results for λ are omitted to save space.

C. Spatial Rate-Limit Algorithm Alone

Figure 6 shows the 5% propagation time of the spatial rate-limit algorithm (alone) with respect to the parameter Φ. The algorithm works well only when both the density of vulnerable hosts ($k \times z$) and the value of Φ are not too large. When $k = 2000$, $z = 5$ and $\Phi = 1000$, the 5% propagation time is 6.3 days. In general, the spatial algorithm should not be used alone.

D. Temporal and Spatial Rate-Limit Algorithms Together

We emphasize that the temporal algorithm and the spatial algorithm are not designed to replace each other. For customer networks with few vulnerable hosts, the temporal algorithm can effectively limit the scan rate because it places the limit on each infected host. In this case, the spatial algorithm is less effective. For customer networks with many vulnerable hosts, the spatial algorithm places a total limit on all infected hosts from a customer network. In this case, the temporal algorithm is less effective. Therefore, the two algorithms are complementary to each other and should be used together.

Table IV shows the 5% propagation time under various conditions when both temporal and spatial algorithms are implemented. Depending on the values of k and z, the propagation time ranges from 5.5 days to 64.7 days.

k	z	$\Phi = 500$	$= 1000$	$= 1500$	$= 2000$	$= 2500$	$= 3000$	$= 3500$	$= 4000$
1000	5	64.7	54.2	50.2	48.6	48.6	48.3	49.2	48.8
2000	5	31.7	26.5	24.8	24.8	24.2	23.9	24.1	23.8
3000	5	21.9	16.7	15.8	15.0	15.2	14.9	15.3	15.2
1000	10	29.1	22.2	20.9	20.6	19.0	18.9	18.8	19.1
2000	10	14.2	9.4	8.6	8.9	8.4	8.8	8.6	8.7
3000	10	8.0	5.8	5.7	5.8	5.5	5.7	5.9	5.8

TABLE IV

5% PROPAGATION TIME (DAYS) FOR "TEMPORAL + SPATIAL"

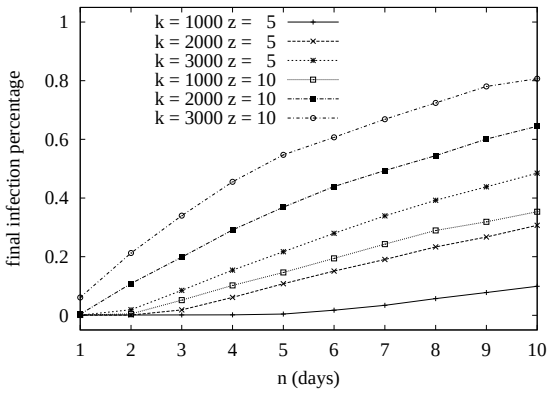

Fig. 7. stop worm propagation by blocking

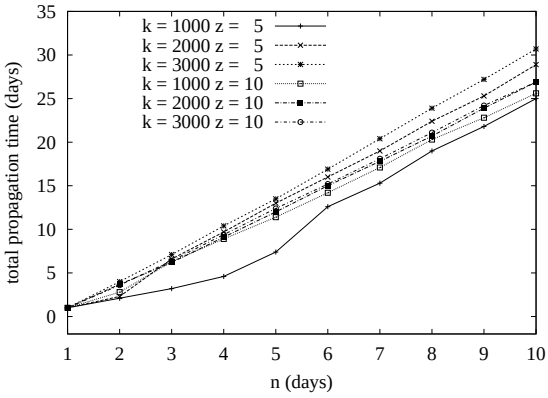

Fig. 8. propagation time before the worm is stopped

To ensure a large propagation time, a very large ISP may partition its customers into multiple defense zones of modest sizes. The rate-limit algorithms can be implemented on the boundary of each zone, consisting of the edge routers to the customer networks of the zone and the internal routers connecting to other zones. In this way, $k \times z$ can be kept modest.

E. DAW

Because DAW blocks persistent scanning sources, it may stop the worm propagation, depending on the value of n. Figure 7 shows the final infection percentage among the vulnerable hosts before all infected hosts are blocked. Even when a large n is selected and the final infection percentage is large, the blocking is still very useful because it considerably slows down the worm propagation as shown in Figure 8. For instance, when $k = 3000$, $z = 10$ and $n = 10$, the final infection percentage is about 80%. However, it will take the worm 27 days to achieve that. Some propagation times are small. For instance, when $n = 1$, the propagation time is just one day. That is because the final infection percentage is extremely small and the worm is quickly blocked out in a day.

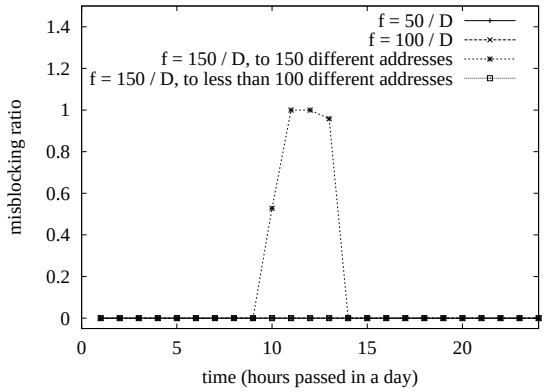

Fig. 9. misblocking ratio

F. Misblocking Legitimate Connection Requests

In order to avoid misblocking legitimate connections made by normal hosts, the values of the DAW parameters should be set larger than the worst-case numbers of a normal host based on traffic measurement, which may be different for the customer networks. If the parameters are set too small, misblocking can happen.

Consider a normal host in a customer network with $\Omega = 100$. We study four cases: (1) the failure rate f of the host is 50 per day, i.e., $f = 50/D$; (2) $f = 100/D$; (3) $f = 150/D$, and all failed connections are made to different addresses that have not been visited by any host in the customer network before; and (4) $f = 150/D$, but the failed connections are made to less than 100 different addresses that have not been visited before. We assume that, after the normal host is blocked, it will stop making failed connections but instead making successful connections to the servers that are known to be up. Figure 9 shows the simulation results. The misblocking ratio stays zero if $f \leq \Omega$ or the number of different addresses to which the host makes failed connections is less than Ω. If $f > \Omega$ and the failed connections are made to different addresses that are not visited before, the host will eventually be blocked (at the 11th hour in the figure). However, if the host stops making failed connections to new addresses, its misblocking ratio will drop back to zero.

From Figure 5, we know that the worm propagation time can be increased by setting Ω smaller. However, if the value of Ω is set too small, misblocking can happen.

G. DAW against External Worm Attack

A more realistic simulation setup should include worm infection within the ISP and from outside. Three neighbor ISPs represent the rest of the Internet. Φ for a neighbor ISP is ten times that of a customer network. Assume the worm initially starts from the Internet and it has infected all vulnerable hosts outside of the ISP. Some vulnerable hosts within the ISP will be infected by the external attackers and then they will try to spread

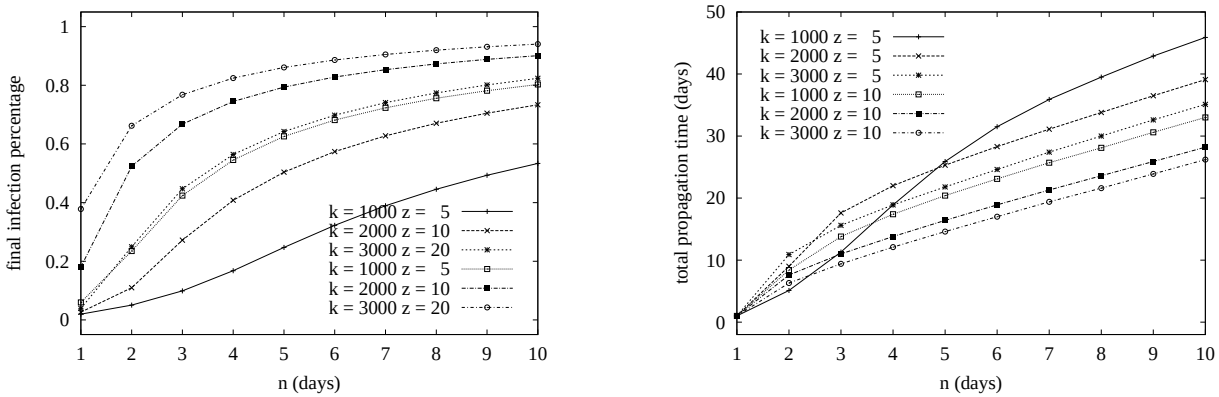

Fig. 10. Inter-ISP worm propagation with respect to n

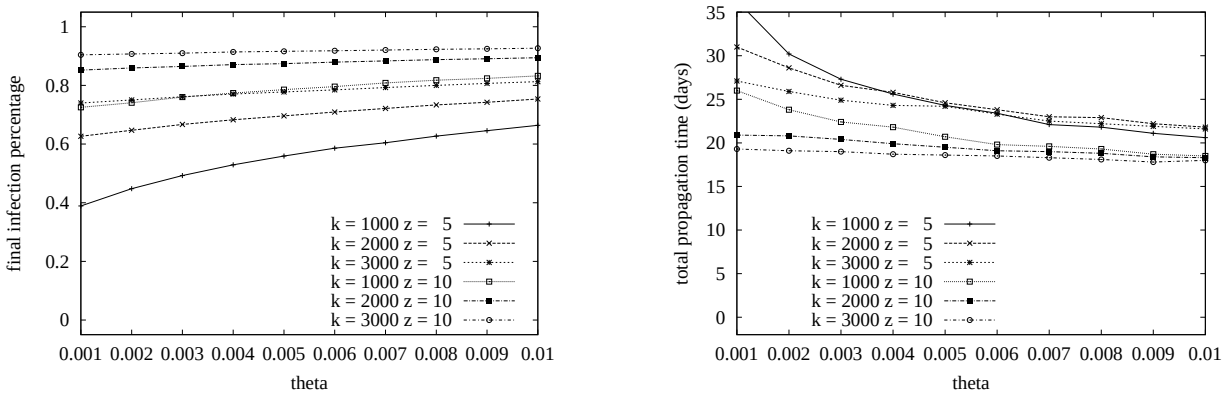

Fig. 11. Inter-ISP worm propagation with respect to θ

the worm internally. The temporal/spatial algorithms (with the aggressive-mode enhancement) are used to rate-limit the external attackers. We show how well DAW controls the infection from both external and (shortly after) internal sources. Note that the goal is not to control the worm propagation on the whole Internet but to control the propagation within the ISP.

The lefthand graph of Figure 10 shows the final infection percentage in the ISP and the righthand graph shows the propagation time before all infected hosts are blocked. While the final infection percentages are quite high when $k \times z$ or n is large, the propagation times are also reasonably large. This is important in practice because, when the news of a widely-spread Internet worm breaks out (less than a day in the case of code red or SQLSlammer), the ISP has the time to take actions when the remedy is published by the Internet security community (e.g., CERT).

Figure 11 repeats the simulation with respect to θ (See Section VI for explanation). The performance of DAW remains well even when θ reaches 1% of the ISP address space.

VIII. Conclusion

This paper proposes a distributed anti-worm architecture (DAW), which integrates a number of new techniques that detect, slow down, and even stop the worm propagation in an internetwork. Our primary goal is to automate the anti-worm defense, which is largely a manual process today. DAW detects possible worm attacks by the edge routers monitoring the local scanning activity and the management station monitoring the global scanning activity. The scanning rate is measured based on the rate of failed connection requests, which sets the worm-infected hosts apart from the normal hosts. DAW ensures sufficient time for human reaction by the use of a temporal rate-limiting algorithm that constrains the maximum scanning speed of any infected host and a spatial rate-limit algorithm that constrains the combined scanning rate of all infected hosts in a network. We evaluate the performance of DAW both analytically and by simulations, which demonstrates that DAW is highly effective in damping the propagation of Internet worms.

References

[1] J. Rochlis and M. Eichin, "With Microscope and Tweezers: The Worm from MIT's Perspective," *Communication of the ACM*, vol. 32, no. 6, pp. 689–698, June 1989.

[2] Computer Emergency Response 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.

[3] eEye Digital Security, "ANALYSIS: CodeRed II Worm," *http://www.eeye.com/html/Research/Advisories/AL20010804.html*, August 2001.

[4] Computer Emergency Response Team, "CERT Advisory CA-2001-26 Nimda Worm," *http://www.cert.org/advisories/CA-2001-26.html*, July 2001.

[5] Computer Emergency Response Team, "CERT Advisory CA-2003-04 MS-SQL Server Worm," *http://www.cert.org/advisories/CA-2003-04.html*, January 2003.

[6] 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.

[7] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton, "StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks," *7th USENIX Security Conference*, January 1998.

[8] D. Wagner, J. S. Foster, E. A. Brewer, and A. Aiken, "A First Step towards Automated Detection of Buffer Overrun Vulnerabilities," *Network and Distributed System Security Symposium*, Feburary 2000.

[9] 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.

[10] 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.

[11] Z. Chen, L. Gao, and K. Kwiat, "Modeling the Spread of Active Worms," *IEEE INFOCOM'03*, March 2003.

[12] 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.

[13] N. Weaver, I. Hamadeh, G. Kesidis, and V. Paxson, "Preliminary Results Using Scale-down to Explore Worm Dynamics," *Proc. of the 2004 ACM Workshop on Rapid Malcode (WORM '2004)*, October 2004.

[14] C. C. Zou, L. Gao, W. Gong, and D. Towsley, "Monitoring and Early Warning for Internet Worms," *Proc. of the* 10^{th} *ACM Conference on Computer and Communication Security (CCS '2003)*, Oct. 2003.

[15] G. Gu, D. Dagon, X. Qin, M. I. Sharif, W. Lee, and G. F. Riley, "Worm Detection, Early Warning, and Response Based on Local Victim Information," *Proc. of the* 20^{th} *Annual Computer Security Applications Conference (ACSAC '2004)*, Dec. 2004.

[16] S. Chen and S. Ranka, "Detecting Internet Worms at Early Stage," *IEEE Journal on Selected Areas in Communications, Special Issue on Recent Advances in Managing Enterprise Network Services*, vol. 23, no. 10, October 2005.

[17] K. Park, H. Kim, B. Bethala, and A. Selcuk, "Scalable Protection against DDoS and Worm Attacks," *DARPA ATO FTN, Technical Report AFRL-IF-RS-TR-2004-100, Dept. of Com. Sci., Purdue University*, 2004.

[18] K. Park, "The Internet As a Large-Scale Complex System, Chapter 1: The Internet as a Complex System," *SFI Studies in the Sciences of Complexity, Oxford University Press, K. Park and W. Willinger (eds.)*, 2005.

[19] M. M. Williamson, "Throttling Viruses: Restricting Propagation to Defeat Malicious Mobile Code," *Proc. of Annual Computer Security Application Conference (ACSAC'02)*, December 2002.

[20] J. Twycross and M. M. Williamson, "Implementing and Testing a Virus Throttle," *Proc. of the* 12^{th} *USENIX Security Symposium (Security '2003)*, Aug. 2003.

[21] T. Liston, "Welcome to My Tarpit: The Tactical and Strategic Use of LaBrea," *Tech. Rep., http://www.threenorth.com/LaBrea/LaBrea.txt*, 2001.

[22] S. Staniford, "Containment of Scanning Worms in Enterprise Networks," *Journal of Computer Security*, 2004.

[23] S. Schechter, J. Jung, and A. W. Berger, "Fast Detection of Scanning Worm Infections," *Proc. of the* 7^{th} *International Symposium on Recent Advances in Intrusion Detection (RAID '2004)*, Sept. 2004.

[24] H. Kim and B. Karp, "Autograph: Toward Automated, Distributed Worm Signature Detection," *Proc. of the* 13^{th} *USENIX Security Symposium (Security '2004)*, Aug. 2004.

[25] S. Singh, C. Estan, G. Varghese, and S. Savage, "The EarlyBird System for Real-time Detection of Unknown Worms," *Proc. of the* 6^{th} *Symposium on Operating System Design and Implementation (OSDI' 2004)*, Dec. 2004.

[26] J. Newsome and D. Song, "Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software," *Proc. of the* 12^{th} *Annual Network and Distributed System Security Symposium (NDSS' 2005)*, Feb. 2005.

[27] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically Generating Signatures for Polymorphic Worms," *Proc. of the 2005 IEEE Symposium on Security and Privacy*, May 2005.

[28] W. Kermack and A. McKendrick, "A Contribution to the Mathematical Theory of Epidemics," *Proc. of the Royal Society of London, Series A*, vol. 115, 1927.

[29] N. Bailey, "The Mathematical Theory of Infectious Diseases," *Oxford University Press, New York*, 1987.

[30] H. W. Hethcote, "The Mathematics of Infectious Diseases," *SIAM Review*, vol. 42, no. 4, pp. 599–653, 2000.

[31] J. O. Kephart and S. R. White, "Directed-Graph Epidemiological Models of Computer Viruses," *Proc. of 1991 IEEE Symposium on Security and Privacy*, May 1991.

[32] D. Moore and C. Shannon and K. Claffy, "Code-Red: A Case Study on the Spread and Victims of an Internet Worm," *Proc. of* 2^{nd} *Internet Measurement Workshop (IMW '2002)*, Nov. 2002.

[33] D. J. Bernstein, "SYN cookies," *http://cr.yp.to/syncookies.html*, 1997.

APPENDIX: PROOF OF THEOREM 2

We first prove that $tokens + c \leq \Omega$ holds for an arbitrary s at any time of the day. It holds initially when the algorithm is activated on s with $tokens = 0$ and $c \leq \Omega/2$. The value of c or $tokens$ changes only after the router receives either a failure reply or a connection request. In the former case, $tokens$ is decreased by one due to the execution of the temporal rate-limit algorithm, and c is increased by one due to the execution of Update_Failure_Rate_Record(). Hence, $(tokens + c)$ stays the same. Now consider the router receives a connection request. The values of $tokens$ before and after receiving the packet are denoted as $tokens\_before$ and $tokens\_after$, respectively. Suppose $tokens\_before + c \leq \Omega$. Based on Lines 6-7, we have

$$\begin{aligned}
&tokens_after \\
&= \min\{tokens_before + \Delta t \times \lambda',\ size\} \\
&\leq tokens_before + \Delta t \times \frac{\Omega\ -\ c\ -\ tokens_before}{\text{the end of the day}\ -\ time} \\
&\leq tokens_before + (\Omega - c - tokens_before) \\
&\leq \Omega - c
\end{aligned}$$

Therefore, $tokens\_after + c \leq \Omega$.

Next we prove that $tokens \geq -rT$ at the end of the day. Consider the case that $tokens < 1$ at the end of the day. Without losing generality, suppose $tokens \geq 1$ before time t_0, $0 \leq tokens < 1$ after t_0 due to the execution of Line 1, and then $tokens$ stays less than one for the rest of the day. After t_0, all connection requests from s are blocked (Line 12). For all requests sent before $t_0 - T$, the failure replies must have already arrived before t_0. There are at most rT requests sent between $t_0 - T$ and t_0. Therefore, there are at most rT failure replies arriving after t_0. We know that $tokens \geq 0$ at t_0. Hence, $tokens \geq -rT$ at the end of the day. Because $tokens + c \leq \Omega$ holds at any time, $c \leq \Omega + rT$ at the end of the day.

The counter c equals the number of failure replies received during the day after the failure-rate record for s is created. Before that, there are at most Ω failure replies counted by the hash-table entry that s maps to. In the worst case all those replies are for s. Therefore, the total number of failure replies for s is no more than $2\Omega + rT$.