Slowing Down Internet Worms

Shigang Chen Yong Tang
Department of Computer & Information Science & Engineering
University of Florida, Gainesville, FL 32611
{sgchen, yt1}@cise.ufl.edu

Abstract

An Internet worm automatically replicates itself to vulnerable systems and may infect hundreds of thousands of servers across the Internet. It is conceivable that the cyber-terrorists may use a wide-spread worm to cause major disruption to our Internet economy. While much recent research concentrates on propagation models, 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. New defense techniques are developed based on behavioral difference between normal hosts and worm-infected hosts. Particulary, a worm-infected host has a much higher connection-failure rate when it scans the Internet with randomly selected addresses. 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. DAW is designed for an Internet service provider to provide the anti-worm service to its customers. The effectiveness of the new techniques is evaluated analytically and by simulations.

1. 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 [4], new worm outbreaks have occurred periodically even though their mechanism of spreading was long well understood. On July 19, 2001, the code-red worm (version 2) infected more than 250,000 hosts in just 9 hours [6]. Soon after, the Nimbda worm raged on the Internet [7]. As recently as January 25, 2003, a new worm called SQL-Slammer [8] reportedly shut down networks across Asia, Europe and the Americas.

There are few answers to the worm threat. One solution is to patch the software and eliminate the security defects [6, 7, 8]. 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. 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.

Most recent research on Internet worms concentrates on propagation modeling [2, 3, 5, 10]. 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 [3]. Williamson proposed to modify the network stack so that the rate of connection requests to distinct destinations is bounded [9]. The main problem is that this approach becomes effective only after the majority of all Internet hosts is upgraded with the new network stack. For an individual organization, although the local deployment may benefit the Internet community, it does not provide the anti-worm protection to its own hosts, whose security depends on the rest of the Internet taking the same action. This gives little incentive for the upgrade without an Internet-wide coordinated effort.

In this paper, we propose a distributed anti-worm architecture (DAW), which is designed for an Internet service provider (ISP) to provide the anti-worm service to its customers. (Note that, from one ISP's point of view, the neighbor ISPs are also customers.) DAW is deployed at the ISP edge routers, which are under a single 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. Particulary, a worm-infected host has a much higher connection-failure rate when it scans the Internet with randomly selected addresses, while 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 minimizes the disturbance to normal users, who generate fewer failed connections than worms. With 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 those parameters on the performance of DAW and use simulations to study the suitable value ranges.

2. Modeling Worm Propagation

The worm propagation can be roughly characterized by the classical simple epidemic model [1, 3, 5].

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

where $i(t)$ is the percentage 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 formly 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,

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

Solving the equation, we have

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

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

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

If $v = 1$, We have

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

Practically it is important to slow down the worm propagation in order to give the Internet community enough time to react. 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 active 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.*

The proof of all theorems in this paper is omitted due to space limitation. □

3. Failure Rate

This paper studies the worms that spread via TCP, which accounts for the majority of Internet traffic. We present a new approach that measures the potential scanning activities by monitoring the failed connection requests.

When a source host makes a connection request, a SYN packet is sent to a destination address. The connection request fails if the destination host does not exist or does not listen on the port that the SYN is sent to. In the former case, an ICMP host-unreachable packet is returned to the source host; in the latter case, a TCP RESET packet is returned. We call an ICMP host-unreachable or TCP RESET packet as a *connection-failure reply* (or simply *failure reply*). The rate of failed connection requests from a host s is called the *failure rate*, which can be measured by monitoring the failure replys that are sent to s.

The failure rate measured for a normal host is likely to be low. For most Internet applications (www, telnet, ftp, etc.), a user normally types domain names instead of raw IP addresses to identify the servers. Domain names are resolved by Domain Name System (DNS) for IP addresses. If DNS can not find the address of a given name, the application will not issue a connection request. Hence, mistyping or stale web links do not result in failed connection requests. An ICMP host-unreachable packet is returned only when the server is off-line or the DNS record is stale, which are both uncommon for popular or regularly-maintained sites (e.g., Yahoo, Ebay, CNN, universities, governments, enterprises, etc.) that attract most of Internet traffic. Moreover, a frequent user typically has a list of favorite sites (servers) to which most connections are made. Since those sites are known to work most

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

of the time, the failure rate for such a user is likely to 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.

On the other hand, the failure rate measured for a worm-infected host is likely to be high. Unlike normal traffic, most connection requests initiated by a worm fail because the destination addresses are randomly picked, which are likely either not in use or not listening on the port that the worm targets at. Consider the infamous code-red worm. Our experiment shows that 99.96% of all connections made to random addresses at TCP port 80 fails. That is, the failure rate is 99.96% of the scanning rate. For worms targeting at software that is less popular than web servers, this figure will be even higher. Let V' is the number of hosts that listen on the attacked port(s). Suppose $V' << N$. The relation between the scanning rate r_s and the failure rate r_f of a worm is

$$r_f = (1 - \frac{V'}{N})r_s \approx r_s \qquad (6)$$

Hence, measuring the failure rate of a worm gives a good idea about its scanning rate. Given the aggressive behavior of a worm-infected host, 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.

4. A Distributed Anti-Worm Architecture

4.1. Objectives

This section presents a distributed anti-worm architecture (DAW), whose main objectives are

- Slowing down (or even stopping) the worm propagation to allow human reaction time. It took the code red just hours to achieve wide infection. Our goal is to prolong that time to tens of days.
- 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. 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.

4.2. Assumptions

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 prevents 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.

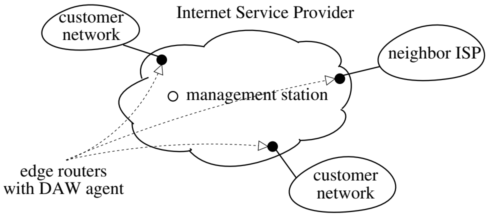

Figure 1: Distributed Anti-Worm Architecturer

We assume that a significant portion of failure replys are not blocked within the ISP. If the ISP address space is *densely* populated, then it is required that a significant portion of TCP RESET packets are not blocked, which is normally the case. If the ISP address space is *sparsely* populated, then it is required that ICMP host-unreachable packets from a significant portion of addresses are not blocked, which can be easily satisfied. Because there are many unused addresses, the ISP routers will generate ICMP host-unreachable for those addresses. Hence, the ISP simply has to make sure its own routers do not filter ICMP host-unreachable until they are counted.

If some customer networks block all incoming SYN packets except for a list of servers, their filtering routers should either generate ICMP host-unreachable for the dropped SYN packets or, in case that ICMP replys are undesirable, send log messages to an ISP log station. Upon receipt of a log message, the log station sends an ICMP host-unreachable towards the sender of the SYN packet. When an ISP edge router receives an ICPM host-unreachable packet from the log station, it counts a connection failure and drops the packet.

4.3. DAW Overview

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 replys sent to the customer network that the edge router connects to. It identifies the potential offending hosts 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 with persistently high failure rates.

Each edge router reads a configuration file from the management station about what source addresses S and what destination ports P that it should monitor and regulate. 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, 23 for telnet, and 21 for ftp. It should exclude the applications that are not suitable for DAW; for example, a hypothetical application runs with an extremely high failure rate, making normal hosts undistinguishable from worms targeting at the application. While DAW is not designed for all services, it is particularly effective in protecting the services whose clients involve human interactions such as web browsering, which makes 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 $m (> 1)$ edge routers with the same ISP, the DAW agent should be stalled on all m edge routers. If some edge routers are with different ISPs that do not implement DAW, the network can be infected via those ISPs but then are restricted in spreading the worm to the customer networks of the ISPs that do implement DAW. For the purpose of simplicity, we do not consider multi-homed networks in the analysis. We discuss the details of DAW below.

4.4. 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( )
$c \leftarrow c + 1$
if ($c$ is a multiple of 100)
    $f' \leftarrow 100/($the current system clock $- t)$
    if ($c = 100$)
        $f \leftarrow f'$
    else
        $f \leftarrow \beta \times f + (1 - \beta) \times f'$
    $t \leftarrow$ the current system clock
```

It is unnecessary to create individual failure-rate records for those hosts that occasionally make a few failed connections. Each 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, it hashes the destination address to a table entry 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. If there are too many records, it retains those with the largest counters. A failure-rate record is removed if its counter c registers too few failed connections in a period of time.

4.5. Basic Rate-Limit Algorithm

If the failure rate of an address s is larger than λ, there must be a failure-rate record created for s because the hash-table entry that s maps to must have a rate exceeding λ, which causes records for individual addresses to be created.

Let F_λ be the set of addresses whose failure rates are larger than λ. For each $s \in F_\lambda$, the router attempts to reduce its 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 $s$
$tokens \leftarrow tokens - 1$

Upon receipt of a connection request from $s$
$\Delta t \leftarrow$ the current system clock $- time$
$tokens \leftarrow \min\{tokens + \Delta t \times \lambda,\ size\}$
$time \leftarrow$ the current system clock
if ($tokens \geq 1$)
    forward the request
else
    drop the request
```

It should be emphasized that the above algorithm is not a traditional token-bucket algorithm that buffers the oversized bursts and releases them at a fixed average rate. The purpose of our algorithm is not to shape the flow of incoming failure replys but to shape the "creation" of the failure replys. It ensures that the failure rate of any address in S stays below λ, which effectively restricts the scanning rate of any worm-infected host (Eq. 6).

This and other rate-limit algorithms are performed on individual addresses. They are not performed on the failure-rate records in the hash table; that is because otherwise many addresses would have been blocked due to one scan source mapped to the same hash-table entry.

One fundamental idea of DAW is to make the speed of worm propagation no longer determined by the worm parameters set by the attackers, but by the DAW parameters set by the ISP administrators. Below we propose more advanced rate-limit algorithms to give the defenders greater control.

4.6. 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. 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. 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 with individual failure-rate records and $\forall s \in F_\Omega$, 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 $s$
$tokens \leftarrow tokens - 1$

Upon receipt of a connection request from $s$
$\Delta t \leftarrow$ the current system clock $- \, time$
if ($c \leq \Omega/2$)
    $tokens \leftarrow \min\{tokens + \Delta t \times \lambda, \ size\}$
else
    $\lambda' \leftarrow \frac{\Omega - c - tokens}{\text{the end of the day } - \, time}$
    $tokens \leftarrow \min\{tokens + \Delta t \times \lambda', \ size\}$
$time \leftarrow$ the current system clock
if ($tokens \geq 1$)
    forward the request
else
    drop the request
```

The temporal rate-limit algorithm constrains both the maximum failure rate and the maximum number of failed requests. 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" ($\Omega - c - tokens$) on the rest of the day, which ensures that connections will be forwarded throughout the day. *Particularly, a host can make successful connections at any rate at any time of the day (e.g., browsing the favorite web sites that are up) because the constraint is on failure replys only.*

Theorem 2 *When the temporal rate-limit algorithm is used, the number of failure replys 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.*

rT is normally small because the typical round trip delay across the Internet is in tens or hundreds of milliseconds. Hence, if $\Omega = 300$, the average scanning rate of a worm is effectively limited to about $2\Omega/D = 0.42/min$. In comparison, Williamson's experiment showed that the scanning rate of the code red was at least 200 / second [9], which is more than 28,000 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.

4.7. 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 replys 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 replys 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 (e.g., 1000) 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.

4.8. Spatial Rate-Limit Algorithm

While each infected host is regulated by the temporal rate-limit algorithm, there may be many of them, whose aggregated scanning rate can be very high. DAW uses a spatial rate-limit algorithm to constrain the combined scanning rate of infected hosts in a customer network. Let Φ be the 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 replys, and activates the spatial algorithm. If there are too many records, it retains those with the largest counters. Let F_Φ ($\in S$) be the set of addresses whose counters exceed a small threshold τ (e.g., 50), which excludes the obvious normal hosts. The spatial rate-limit algorithm is the same as the temporal algorithm except that s, Ω, and c are replaced respectively by F_Φ, Φ, and the total number of failure replys to F_Φ received after the spatial algoirthm is activiated.

For any address s in $F_\Omega \cap F_\Phi$, the temporal rate-limit algorithm is first executed and then the spatial rate-limit algorithm is executed. The reason to apply the temporal algorithm is to prevent a few aggressive infected hosts from keeping reducing $tokens$ to zero. On the other hand, if there are a large number of infected hosts, which cause the spatial algorithm to drop most requests, then the router should temporarily block the addresses whose failure-rate records have the largest counters.

The edge routers may be configured independently with some running both the temporal and spatial algorithms but some running the temporal algorithm only. For example, the edge routers for the neighbor ISPs should have large Φ values or not run the spatial algorithm.

Theorem 3 *When the spatial rate-limit algorithm is used, the total number of failure replys 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.*

$mr'T$ is likely to be small because both r' and T are small. The following analysis is based on a simplified model. A more general model will be used in the simulations. Suppose there are k customer networks, each with V/k vulnerable hosts. Once a vulnerable host is infected, we assume all other vulnerable hosts in the same customer networks 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 percentage of customer networks that are infected by the worm. Following a similar process that derives Eq. 2, we have

$$\frac{dj(t)}{dt} = \frac{2V\Phi}{ND} j(t)(1 - j(t))$$
$$j(t) = \frac{e^{\frac{2V\Phi}{ND}(t-T)}}{1 + e^{\frac{2V\Phi}{ND}(t-T)}}$$

Assume there is one infection at time 0. We have $T = -\frac{ND}{2V\Phi}\ln\frac{1}{k-1}$. 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.

4.9. 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. If the worm-infected hosts perform high-speed scanning, they will each be blocked out after n days of activity. Hence the worm propagation may be stopped before an epidemic materializes, according to Eq. (5).

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. 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 number of infected networks is bounded by

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

On the other hand, 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 underestimated 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 relys on a different approach to address this problem. During each day, an edge router reports the total number of connection requests and the total number of failure replys to the management station, which watches for the global trend of scan activities and takes actions accordingly. The details are omitted due to space limitation.

4.10. Warhol Worm and Flash Worm

The Warhol worm and the Flash worm are hypothetical worms studied in [5], which embodied 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 [5]. With DAW, that will take about $N/2\Omega$ days. Suppose $\Omega = 300$ and $N = 2^{32}$. That would be 19611 years. 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. Considering the size of IPv4 is $2^{32} \approx 4.3 \times 10^9$, 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 $200/\text{second}$, it takes less than 36 minutes for 10000 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 requires a prescan of the entire Internet before it is released. Such a prescan takes too long under DAW. In addition, each infected host can only scan about 2Ω addresses per day, which limits the propagation speed of the worm if L is large.

4.11. Forged Failure Replys

To prevent forged failure replys 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 4.3. Each table entry contains a source address, a source port, a destination address, and a destination port, identifying a connection request. Only those failure replys 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 ($c := c + 1$) only when the invariant is not violated.

5. Simulation

We use simulations to evaluate the performance of DAW. Figure 2 shows how the rate-limit algorithms slow down the worm propagation. The simulation parameters are given as follows. $\lambda = 1$/sec. $\Omega = 300$. $\Phi = 3000$. $n = 7$ days. The number of customer networks are $k = 10000$. The average number of vulnerable hosts per customer network is $z = 10$. The numbers of vulnerable hosts in different customer networks follow an exponential distribution, suggesting a scenario where most customer networks have ten or less public servers, but some have large numbers of 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 shortly.

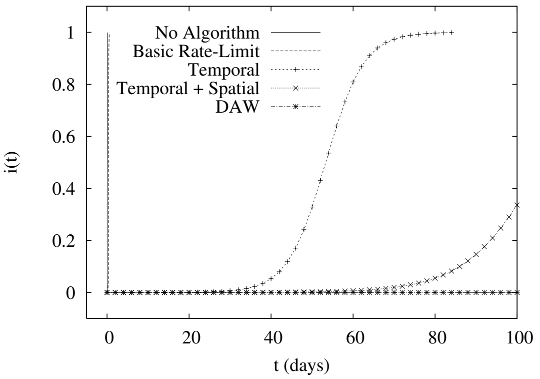

Figure 2: worm-propagation comparison

Figure 2 compares the percentage $i(t)$ 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 request rates, and the spatial rate-limit algorithm is applied only on the hosts whose failure counters exceed a threshold $\tau = 50$. The shape of the curve for "No Algorithm" depends on the worm's scanning rate, which is 10/sec in the 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 80 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 $i(t) = 0.000034$.

Table 1 shows the time it takes the worm to infect 5% of vulnerable hosts (called *5% propagation time*) under various conditions with Temporal + Spatial implemented. Depending on the size (k and z) of the ISP, the propagation time ranges from 10.0 days to 350.3 days. To ensure a large propagation time, a very large ISP may partition its customers into multiple defense zones of modest sizes. DAW can be implemented on the boundary of each zone, consisting of the edge routers to the customer networks of the zone and the inter-

k	z	$\Omega = 1000$	$= 3000$	$= 5000$	$= 7000$
5000	10	350.3	116.8	69.6	50.2
5000	20	237.2	79.1	47.2	33.9
10000	10	190.1	63.5	38.1	27.1
10000	20	127.9	42.5	25.5	18.3
20000	10	103.3	34.2	20.6	14.6
20000	20	68.9	22.9	13.8	10.0

Table 1: 5% propagtion time (days) for "Temporal + Spatial"

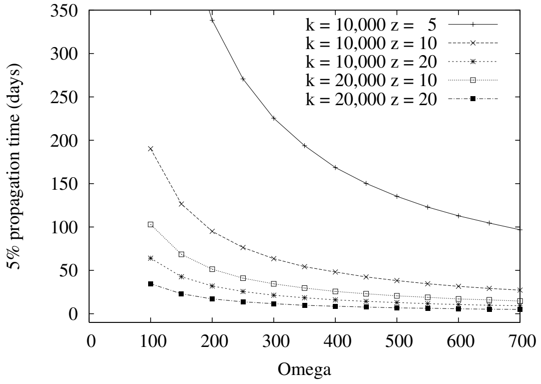

Figure 3: effectiveness of the temporal rate-limit algorithm

nal routers connecting to other zones.

Figure 3 shows the performance of the temporal rate-limit algorithm with respect to the parameter Ω. As expected, the propagation time decreases when Ω increases. The algorithm performs very well for modest-size ISPs (or zones). When $k = 10000$, $z = 10$ and $\Omega = 3000$, the 5% propagation time is 63.6 days. Figure 4 shows the performance of the spatial rate-limit algorithm (alone) with respect to the parameter Φ. The algorithm works well for modest-size ISPs (or zones) even for large Φ values. When $k = 10000$, $z = 10$ and $\Phi = 7000$, the 5% propagation time is 27.2 days. The performance of the two algorithms is comparable when $\Phi = z \times \Omega$, where the total temporal rate limit of the local infected hosts is equal to the spatial rate limit. As shown in the figures, if $\Phi > z \times \Omega$, the temporal algorithm works better; if $\Phi < z \times \Omega$, the spatial algorithm works better. Therefore, the two algorithms are complementary to each other and they are both adopted by DAW.

6. 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 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.

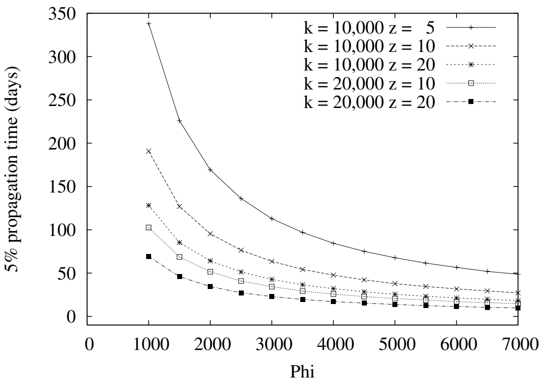

Figure 4: effectiveness of the spatial rate-limit algorithm

References

[1] H. W. Hethcote. The Mathematics of Infectious Diseases. *SIAM Review*, 42(4):599–653, 2000.

[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] 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.

[4] J. Rochlis and M. Eichin. With Microscope and Tweezers: The Worm from MIT's Perspective. *Communication of the ACM*, 32(6):689–698, June 1989.

[5] 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.

[6] 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.

[7] C. E. R. Team. CERT Advisory CA-2001-26 Nimda Worm. *http://www.cert.org/advisories/CA-2001-26.html*, July 2001.

[8] C. E. R. Team. CERT Advisory CA-2003-04 MS-SQL Server Worm. *http://www.cert.org/advisories/CA-2003-04.html*, January 2003.

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