Inside the Permutation-Scanning Worms: Propagation Modeling and Analysis

Parbati Kumar Manna, *Member, IEEE*, Shigang Chen, and Sanjay Ranka, *Fellow, IEEE*

***Abstract*—In recent years, both sophistication and damage potential of Internet worms have increased tremendously. To understand their threat, we need to look into their payload for signatures as well as propagation pattern for Internet-scale behavior. An accurate analytical propagation model allows us to comprehensively study how a worm propagates under various conditions, which is often computationally too intensive for simulations. More importantly, it gives us an insight into the impact of each worm/network parameter on the propagation of the worm. Traditionally, most modeling work in this area concentrates on the relatively simple random-scanning worms. However, modeling the permutation-scanning worms, a class of worms that are fast yet stealthy, has been a challenge to date. This paper proposes a mathematical model that *precisely* characterizes the propagation patterns of the general permutation-scanning worms. The analytical framework captures the interactions among all infected hosts by a series of interdependent differential equations, which are then integrated into closed-form solutions that together present the overall worm behavior. We use the model to study how each worm/network parameter affects the worm propagation. We also investigate the impact of dynamic network conditions on the correctness of the model.**

***Index Terms*—Network security, worm modeling.**

I. Introduction

COMPUTER worms interest security analysts immensely due to their ability to infect millions of computers in a very short period of time [1]. In recent years, both sophistication and damage potential of worms have increased tremendously [2]–[4]. Symantec stated in their 2008 global Internet security threat report [5], "Of the top 10 new malicious code families detected in the last six months of 2007, five were Trojans, two were worms, two were worms with a back door component, and one was a worm with a virus component."

In order to counter the threat, we need to look into their content (for signatures) as well as propagation pattern (for Internet-scale behavior) [6]–[11]. The propagation characteristics of a worm shows what kind of network traffic will be generated by that worm and how fast the response time must be for countermeasures. Therefore, in order to understand (and possibly counter) the damage potential of worms, it is very important to characterize their overall propagation properties.

Although modeling worm propagation has been an active research area [12]–[16], one might question the practical importance of such work if it is possible to obtain fairly good approximation of the worm's propagation characteristics by running a simulator for a sufficient number of times and taking the average. However, there are reasons why simulations may not always be able to produce the intended results. First, it often takes a long time—16 h in our case on a Intel Xeon 2.80-GHz processor for 400 M hosts that are estimated to be in today's IPv4 space—to simulate a single run of worm propagation for one set of worm/network parameters. To learn the average behavior, many such runs need to be performed, and the whole simulation process has to be redone for any parameter change, e.g., for a different population size of vulnerable hosts or a different scanning speed of infected hosts. Second, the simulation overhead can be prohibitively high in some cases. Suppose we want to simulate a worm that exploits a commonly used Windows service on today's Internet. It means that the vulnerable population size could be in the order of several hundred millions as Windows machines predominate in the Internet. If there are 300 M such computers, they will entail 300 M records in the simulation, one for each vulnerable host. Even if each record is one integer (keeping its address alone), it will require a memory of 1.2 GB. Now, if we want to study the impact of migration from IPv4 to IPv6 on worm propagation, a full-scale simulation of scanning the address space of size 2^{128} will be computationally infeasible. In comparison, numerical computation based on a mathematical model takes little time to produce the accurate propagation curves. Third, simulation results themselves do not always give the *mathematical insight* that a formal model provides. One may guess upon the impact of various parameters on worm propagation based on extensive simulations (which may take enormous time), but such guesses can never be as precise and comprehensive as an analytical model, which tells exactly why and by how much a parameter change will affect the outcome.

Traditionally, most modeling work [12]–[15] concentrates on the relatively simple random-scanning worms, which scan the Internet either randomly or with bias toward local addresses in order to reach all vulnerable hosts. This strategy leaves a large footprint on the Internet (which reveals the worm's presence), and different infected hosts may end up scanning the same address repeatedly. In recent years, worm technologies have advanced rapidly to address these problems. By enabling close coordination among all infected hosts, the permutation-scanning worms (introduced in the seminal paper [12] by Staniford *et*

Manuscript received October 27, 2008; revised July 02, 2009; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor P. Van Mieghem.

P. K. Manna was with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611 USA. He is now with the Security Center of Excellence, Intel, Hillsboro, OR 97124 USA.

S. Chen and S. Ranka are with the Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL 32611 USA (e-mail: sgchen@cise.ufl.edu).

Digital Object Identifier 10.1109/TNET.2009.2034655

al.) minimize the overall traffic volume for scanning the Internet through a divide-and-conquer approach. There, each active infected host is responsible for scanning a subset of all addresses, and this subset may vary over time. Such a cooperation strategy empowers the worm with the ability to propagate either much faster—or, alternatively, much stealthier (if the infected hosts scan at lower rates). Warhol worms, which are similar to permutation-scanning worms with larger hitlists, have been shown using simulations to be able to infect the whole Internet in a matter of minutes [12]. However, understanding these potent worms through mathematical modeling has remained a challenge to date.

In this paper, we propose a mathematical model that *precisely* characterizes the propagation patterns of the permutation-scanning worms. The analytical framework captures the interactions among all infected hosts by a series of interdependent differential equations, which together describe the overall behavior of the worm. We then integrate these differential equations to obtain the closed-form solution for worm propagation. We use simulations to verify the numerical results from the model and show how the model can be used to assess the impact of various worm/network parameters on the propagation of permutation-scanning worms. We also investigate the impact of dynamic network conditions on the correctness of the model, considering network congestion, bandwidth variability, Internet delay, host crash, and patch.

The rest of this paper is organized as follows. Section II describes the permutation-scanning worms. Section III introduces several important concepts underlying our mathematical model. Sections IV and V present the exact propagation models for the basic permutation-scanning worm and its general extension, respectively. Section VI derives the closed-form solution. Sections VII and VIII discuss how different worm/network parameters and real-life network constraints will affect the worm propagation, respectively. Section IX draws the conclusion.

II. Anatomy of Permutation-Scanning Worms

We first describe the divide-and-conquer nature of the permutation-scanning worms. We then explain the reason for address permutation and discuss the stealthy potential of such worms.

A. Divide-and-Conquer

To avoid repeatedly scanning the same addresses, the infected hosts may collaborate by dividing the IPv4 address ring into disjoint sections, each of which will be scanned by one host. Each initially infected host starts "walking" along the address ring clockwise from its own location and *sequentially* scans the traversed addresses. Whenever it infects a host, it continues walking and scanning the addresses after that host, while the newly infected host performs a *jump*, i.e., chooses a random location on the ring and starts to walk and sequentially scan addresses clockwise after that location. The reason for this jumping action is that if the newly infected host instead started scanning sequentially after its own address, then those addresses would get scanned by both this host and its infector—a needless duplication of the same work. Now, if the scan performed by a host h_1 hits an already-infected host h_2, h_1 knows that addresses after h_2 must have already been scanned by another infected host that infected h_2, or by h_2 itself in case h_2 was one of the initially infected hosts to begin with. In either case, it is unproductive for h_1 to continue scanning addresses after h_1. Therefore, h_1 *jumps* to a random location on the ring and starts to scan addresses clockwise after that location. An infected host retires (stops scanning) after hitting a certain number of already-infected hosts.

An alternative to the above random-jump approach is to assign each infected host an exclusive section of the address ring for scanning. As a host sequentially scans its section, when it infects another host, it assigns half of the remaining unscanned addresses to the latter and adjusts its section boundary accordingly. When a host reaches the end of its section, it retires. The problem with this approach is that it is not fault-tolerant. If one infected host is blocked out or somehow crashes, it may leave many addresses in its section unscanned. Random jumps by infected hosts before they retire (as described previously) help solving this problem by providing redundancy, and this paper will focus on such worms only.

B. Permutation

While the above divide-and-conquer method reduces the chance of scanning the same address again and again, it has a serious weakness. Since the IP addresses scanned by an infected host are contiguous, it is susceptible to be identified by address-scan detectors or other IDSs that look for worms performing local subnet scanning. To counter this, Staniford *et al.* [12] show that a worm can permute the IP address space into a virtual one (called the *permutation ring*) through encryption with a key. The divide-and-conquer method is then applied on this permutation ring. While each infected host logically goes through contiguous addresses on the permutation ring, it actually scans the IP addresses that the permuted addresses are decrypted to, which cannot be easily picked up by address-scan detectors because those IP addresses are pseudo-random and distributed all over the Internet.

C. Stealth

Fast propagation and stealth are two conflicting goals that the worm designers strive to balance. To spread fast, infected hosts should scan at high rates, which however makes them easier to be detected [1], [7], [8]. To be stealthy, they have to act as normal as possible by scanning the Internet at a controlled low rate, which is a parameter that can be set before worm release. A stealthy worm can be more harmful. A fast worm generates headline news, such as Slammer [1], which caused widespread network congestion across Asia, Europe, and the Americas. However, such a worm is more likely to be detected quickly and attract defense resources for its elimination. A stealthy worm propagates slower, but may stay undetected for a long time, potentially doing more harm.

Permutation-scanning worms are particularly suited for stealth due to their divide-and-conquer nature. If all infected hosts scan at a low rate (such as one address every few seconds) in order to achieve stealth, then the impact of the network conditions on the worm propagation can become negligible because most hosts on the modern Internet are likely to have the bandwidth for delivering one scan packet every few seconds.

In this case, our mathematical modeling can largely ignore the impact of dynamic network conditions. However, when the infected hosts scan at higher rates, such impact should be considered, as we will do in Section VIII.

D. Hitlist

The initial part of worm propagation is most time-consuming, as only a few infected hosts perform scanning in a vast address space. Once the number of infected reaches a critical mass, the rate of new infections goes up drastically. To improve the initial scanning speed of a stealthy worm, one can use a *hitlist* as proposed in [12], which is a precompiled list of target addresses that are very likely to be vulnerable, e.g., a list of hosts with port 80 open for a worm targeting at a certain type of Web servers. During the hitlist-infection phase, the very first infected host scans the IP addresses in the hitlist, and whenever it infects one, it gives away half of the remaining hitlist to the newly infected host so that together they can infect all hosts in the original hitlist quicker. This process repeats, and as a result, if v out of the S addresses in the hitlist turn out to be vulnerable hosts, all those hosts will get infected in $O(\frac{S}{vr}\log_2 v)$ time, where r is the scanning rate. Even for a modestly big hitlist, this time is miniscule compared to the time it will take to infect the rest of the vulnerable hosts outside the hitlist. To illustrate with an example, suppose there are about 1 M vulnerable hosts in IPv4 and a worm starts with a hitlist of $S = 10$ K hosts, with approximately $v = 5$ K of them actually being vulnerable. If the scanning rate r is 1000 addresses/s, then the time taken to infect the initial 5 K hosts in the hitlist will be approximately 0.025 s, which can arguably be ignored compared to the time the worm will take to infect the rest of the vulnerable hosts in the Internet. Thus, to keep the model simple, if the hitlist contains v vulnerable hosts, we assume that all v of them are infected at time $t = 0$.

III. Scanzone and Classification of Vulnerable Hosts

We introduce the concept of *scanzone* and classify vulnerable hosts into different categories, which lays the foundation for our analysis in the next section.

A. Terminology and Notations

We classify infected hosts into two categories: 1) *active infected hosts*, which are actively scanning for vulnerable hosts; and 2) *retired infected hosts*, which have stopped scanning. When the context makes it clear, we omit "infected" from the above terms. Other terms are defined as follows:

Jump: When an infected host chooses a random location on the permutation ring to perform its sequential scan along the ring, we say that the host *jumps* (to that location).

Old Infection: When an active host hits a vulnerable host h that was infected previously, we denote the event (as well as host h) as an *old infection*.

New Infection: When an active host hits a vulnerable host h that was *not* previously infected, we denote the event (as well as host h) as a *new infection*.

***k*-Jump Worm**: A permutation-scanning worm is called a k-jump worm if an active host, upon hitting an old infection, jumps to a new location on the permutation ring to resume scanning, but it will retire when hitting its $(k+1)$-th old infection. When a vulnerable host not in the hitlist becomes a new infection, it jumps to a random location on the ring to begin its scan. Subsequently this host can make k other jumps after hitting old infections on the ring. For a vulnerable host in the hitlist, it begins scanning from its own location, and then it can make k jumps. To summarize, irrespective of whether it was in the hitlist or not, a host in a k-jump worm is allowed to jump for its first k old infections, but when it hits its $(k+1)$-th old infection, it retires.

0-Jump Worm: A permutation-scanning worm is called a 0-jump worm if an active host retires upon hitting its very first old infection. It is a special case of k-jump worm with $k = 0$. A vulnerable host not in the hitlist can make one jump when it becomes a new infection itself, but subsequently when it hits an old infection, it will retire immediately.

B. Scanzone of an Active Infected Host

As an active infected host h scans the addresses along the permutation ring, it leaves behind a contiguous segment of scanned addresses. This contiguous segment, called the *scanzone* of host h, contains the addresses that h has scanned since its last jump, or time 0 if h has not jumped yet; it may contain more addresses if scanzone merge happens, which will be discussed shortly. The scanzones of all active hosts cover all addresses scanned so far. The address of each infected host belongs to a scanzone because it is a scanned address. The *front end* of a scanzone is the address that h is currently scanning; the *back end* refers to the address at the other end of the scanzone, which is the first address that h scans after its last jump. Evidently, all vulnerable hosts in a scanzone must have been infected. Among all infected hosts in a scanzone, the one that is closest to the back end is called the *tail* of the scanzone, and the one that is closest to the front end is called the *head* of the scanzone. The portion of a scanzone between the tail and the head is referred to as the *covered area* (portrayed as ▪▪▪▪▪ in Fig. 1) of the scanzone. A scanzone may not have a tail (or head) if the active infected host has not hit any vulnerable host since its last jump, and it may not have any covered area if it does not have at least two infected hosts in it.

As h scans more and more addresses, the front end advances to expand the scanzone. However, when h hits an old infection h_{old} (which must belong to the scanzone of some active infected host h_1), h surrenders its scanzone by merging it to h_1's scanzone. Then, h jumps to a random location to create its new scanzone afresh, or retires if h_{old} is the $(k+1)$-th old infection that it hits. Therefore, the back end of a scanzone may also change if the front end of another scanzone catches up its tail and causes a merge. Merges create larger scanzones. Eventually, all scanzones will be merged into one when all active hosts retire. Only active hosts have scanzones (uninfected or retired hosts do not).

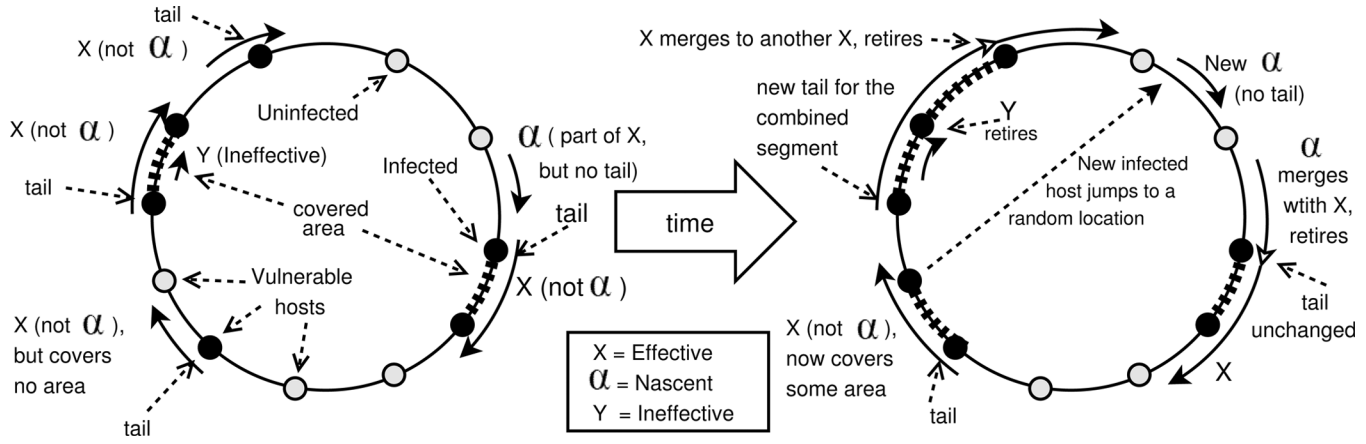

Fig. 1. Depiction of scanzones for a 0-jump worm over time. Scanzones of active hosts are depicted as arcs on the permutation ring. Uninfected and infected vulnerable hosts are depicted as white and dark dots on the permutation ring, respectively.

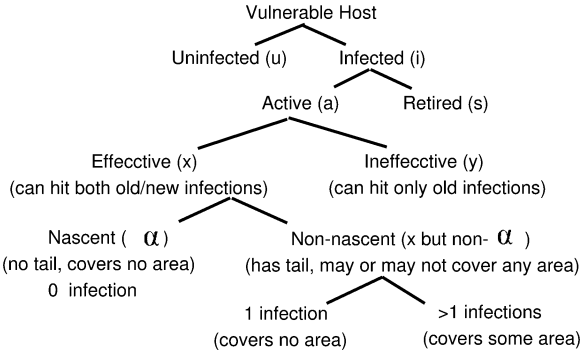

Fig. 2. Classification of vulnerable hosts for a permutation-scanning worm.

We stress that an infected host does not need to know its scanzone; it is an abstract concept used in our mathematical modeling only. The scanzones are shown as arcs on the permutation ring in Fig. 1, which also illustrates other concepts to be defined in this section.

C. Classification of Vulnerable Hosts

In our model, we define classes u, i, a, s, x, y, α for vulnerable hosts that are uninfected, infected, active, retired, effective, ineffective, and nascent, respectively, and we deliberately make the class notations the same as the corresponding variables in our later propagation model for the *sizes* of these classes. Fig. 2 shows the containment relationship among different classes.

While other classes are self-explaining, we focus on classifying the active hosts, class a, into subcategories, class x and class y, based on whether an active node's scanning is effective or not, i.e., whether it has the potential to generate new infection *before* hitting an old one (note that since the size of the ring is finite, every active host will eventually hit an old infection).

- **Ineffective hosts (class y)**: An active infected host is considered *ineffective* if it is impossible for the host to generate any new infection in the future before hitting an old infection. An active host that jumps into a covered area to begin its scanning is evidently ineffective since its first hit will always be an old infection.
- **Effective hosts (class x)**: An active infected host is considered effective if it can *potentially* generate a new infection in the future *before* it hits an old one. When an infected host jumps to a point outside of any covered area and starts scanning from that point on, it can potentially generate new infections and is thus called *effective*. The effective hosts are branded as class x. This class is further subdivided as follows:
  - **Nascent hosts (class α)**: The effective hosts that are yet to infect any vulnerable host in their *current* scanzones (which, obviously, have no head or tails) are termed as *nascent* (class α). An active host becomes nascent after it takes a jump and lands outside any covered area. Note that the host starts with a fresh scanzone after the jump.
  - **Non-nascent hosts**: Once a nascent host hits a new infection, it becomes a non-nascent effective host; the host it just infected becomes the tail of its scanzone. Also, each of the initially infected hosts starts as a non-nascent effective host because its scanzone has a tail from the very beginning (the active host itself).

We observe that every infected host in the address space belongs to the scanzone of a non-nascent effective host. This is true at the beginning as each of the initially infected hosts belongs to its own scanzone. When a non-α effective host h_1 infects another host h_{new}, the address h_{new} becomes part of h_1's scanzone. When h_1 retires by hitting h_{old} (tail of a non-α effective host h_2's scanzone), h_1's scanzone merges with h_2's scanzone, and the infections made in h_1's scanzone now become part of h_2's scanzone. Continuing this way, every infected host remains part of the scanzone of a non-nascent effective host until the last active host retires. It should be noted that the scanzones of nascent or ineffective hosts do not contain any infected hosts.

Fig. 3 gives the class transition diagram for a 0-jump worm. A vulnerable host becomes infected when it is scanned by another infected host. When it jumps, it may be either effective or ineffective, depending on whether it jumps into a covered area or not. An effective host begins as a nascent one and becomes non-nascent once it infects another host. An active host retires upon hitting an old infection. Fig. 1 also provides illustration for transitions among different classes.

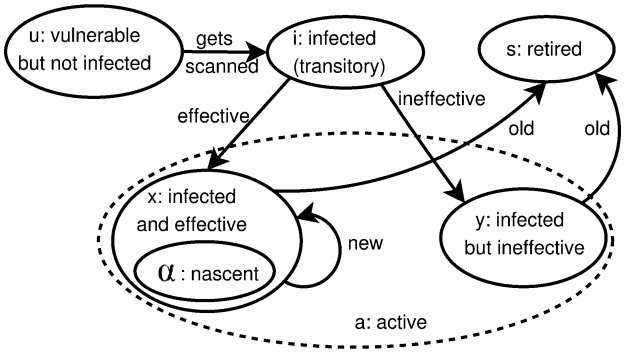

Fig. 3. Class transition diagram of a 0-jump worm. Here, "new" or "old" indicates the event of a *new* or *old* infection. Similarly, "ineffective" or "effective" indicates whether the newly infected host, after the random jump, lands in a covered area or not.

IV. Modeling the Propagation of 0-Jump Worms

In this section, we derive a series of differential equations that together form the propagation model of 0-jump worms. We extend it for k-jump worms in the next section.

A. Important Quantities in Modeling

The propagation model of a worm reflects the fractions of vulnerable hosts that are infected, active, and retired over time. A scan message that does not hit any vulnerable host does not change these numbers. Thus, modeling should only be based on the event of a scan message hitting a vulnerable host. When that event happens, all aforesaid numbers change. We derive the model by analyzing the precise amounts by which they change. To model a 0-jump worm mathematically, we have to compute the following quantities:

Q1: Between time t and $t+dt$ (for an infinitesimally small dt), how many vulnerable hosts will an active host hit with its scan messages?

Q2: When an effective host hits a vulnerable host h, what is the probability that h is an old infection, and what is the probability that h is a new infection? Note that an ineffective host, by definition, never hits a new infection.

Q3: After a newly infected host jumps, what is the probability for it to be ineffective, and what is the probability for it to be effective?

B. Determining the Quantities Using Probabilistic Approach

Let N be the size of the address space, V the total number of the vulnerable hosts, r the scanning rate, and v the number of the vulnerable hosts in the hitlist of a permutation worm.

We use $u(t), i(t), a(t), s(t), x(t), y(t)$, and $\alpha(t)$ to denote the fractions of vulnerable host population that are uninfected, infected, active, retired, effective, ineffective, and nascent at time t, respectively. From Fig. 2, it is easy to see that $u(t)+i(t)=1, i(t)=a(t)+s(t)$, and $a(t)=x(t)+y(t)$.

Answer for Q1: Let $\boldsymbol{f}_{\rm hit}$ be the number of vulnerable hosts that an active host is expected to hit during a period of dt after time t. Since vulnerable hosts are uniformly distributed on the permuted address space due to randomization of the permutation process, every address on the permutation ring has a probability of $\frac{V}{N}$ to be a vulnerable host. An active host scans $r\times dt$ addresses during dt period. Hence, we have $f_{\rm hit}=r\times dt\times\frac{V}{N}$. Note that the vulnerable hosts that are hit may include both new and old infections.

Answer for Q2: When an effective host hits a vulnerable host, let $f_{\rm new}(t)$ ($f_{\rm old}(t)$) denote the probability for the vulnerable host to be a new (old) infection. We observe that an effective host can hit only two types of vulnerable hosts: 1) those that are uninfected; and 2) infected ones that are the *tails* of scanzones for non-α effective hosts. Recall that scanzones of nascent or ineffective hosts do not have tails. At time t, there are $V(1-i(t))$ uninfected vulnerable hosts (possible new infections) and $V(x(t)-\alpha(t))$ tails (possible old infections). Hence, the chance for hitting a new infection is $f_{\rm new}(t)=\frac{V(1-i(t))}{V(1-i(t))+V(x(t)-\alpha(t))}=\frac{(1-i(t))}{(1-i(t))+(x(t)-\alpha(t))}$, and $f_{\rm old}(t)=1-f_{\rm new}=\frac{(x(t)-\alpha(t))}{(1-i(t))+(x(t)-\alpha(t))}$.

Answer for Q3: After a newly infected host jumps to a random location to begin its scanning, let $f_{\rm ineff}(t)$ ($f_{\rm eff}(\boldsymbol{t})$) be the probability for the host to be ineffective (effective). Since a host becomes ineffective when it jumps into a covered area, $f_{\rm ineff}(t)$ must be equal to the fraction of the permutation ring that all covered areas together represent. Because vulnerable hosts are distributed randomly on the ring, it must also be equal to the fraction of vulnerable hosts that are located in the covered areas, excluding tails because, if we use the number of vulnerable hosts in a covered area to represent its length (in a statistical sense), we cannot count both head and tail that delimits the two ends of the area.[1] All infected hosts, $Vi(t)$ of them, are located in the covered areas, and there are $V(x(t)-\alpha(t))$ tails because every non-nascent effective host has a scanzone with a tail by definition. Therefore, $f_{\rm ineff}(t)=\frac{Vi(t)-V(x(t)-\alpha(t))}{V}$, and $f_{\rm eff}(t)=1-f_{\rm ineff}(t)$.

C. Propagation Model

We now derive how $i(t), a(t), s(t), x(t), y(t)$, and $\alpha(t)$ change over time t. Below, we compute the amounts $di(t), da(t), ds(t), dx(t), dy(t)$, and $d\alpha(t)$, by which they change respectively over an infinitesimally small dt after time t. This will give us a set of differential equations that together characterize the propagation of 0-jump worms.

- $di(t)$: This, when multiplied by V, represents the total number of new infections generated during dt. Only effective (class x) hosts can hit new infections. The number of vulnerable hosts hit by effective hosts over dt is $Vx(t)f_{\rm hit}$, and each of them has a probability of $f_{\rm new}(t)$ to be a new infection. Hence, $di(t)=x(t)f_{\rm hit}f_{\rm new}(t)$.
- $dx(t)$: Each of the $x(t)f_{\rm hit}f_{\rm new}(t)V$ new infections has a probability of $f_{\rm eff}(t)$ to be effective. This adds $x(t)f_{\rm hit}f_{\rm new}(t)Vf_{\rm eff}(t)$ new effective hosts after dt. On the other hand, effective hosts together hit $x(t)f_{\rm hit}f_{\rm old}(t)V$ old infections during dt, each causing an effective host (that hits the old infection) to retire. Combining the above two numbers and representing the gross change in fraction, we have $dx(t)=x(t)f_{\rm hit}f_{\rm new}(t)f_{\rm eff}(t)-x(t)f_{\rm hit}f_{\rm old}(t)$.
- $d\alpha(t)$: Each nascent host (which is effective by definition) is no longer nascent once it hits any vulnerable host.

[1] A scanzone with a single infection can be thought of as having a covered area of length 0.

Each of its $r \times dt$ scan messages has a $\frac{V}{N}$ probability of hitting a vulnerable host. Hence, the probability for a nascent host to become non-nascent over dt is $r \times dt \times \frac{V}{N} = f_{\text{hit}}$ because, as dt approaches to zero, the joint probabilities for two or more hits are negligible. This reduces the number of nascent hosts by $\alpha(t)Vf_{\text{hit}}$. On the other hand, since all new effective hosts created during dt start as nascent, we have $x(t)Vf_{\text{hit}}f_{\text{new}}(t)f_{\text{eff}}(t)$ new nascent hosts. Combining these two numbers and representing the gross change in fraction, we have $d\alpha(t) = x(t)f_{\text{hit}}f_{\text{new}}(t)f_{\text{eff}}(t) - \alpha(t)f_{\text{hit}}$.

- $\boldsymbol{dy(t)}$: Recall that whenever a host jumps into a covered area, it becomes ineffective. For a 0-jump worm, only the newly infected hosts make a jump and thus only they may increase $y(t)$. There are $x(t)Vf_{\text{hit}}f_{\text{new}}(t)$ new infections, and each has a probability of $f_{\text{ineff}}(t)$ to become ineffective. On the other hand, when an existing ineffective host hits a vulnerable host, it retires since ineffective hosts can hit old infections only. Combining these two factors and representing the gross change in fraction, we have $dy(t) = x(t)f_{\text{hit}}f_{\text{new}}(t)f_{\text{ineff}}(t) - y(t)f_{\text{hit}}$.
- $ds(t)$: Whenever an effective host hits an old infection, or an ineffective host hits *any* vulnerable host (which must be an old infection), it retires. Within time dt, there are $x(t)Vf_{\text{hit}}f_{\text{old}}(t) + y(t)Vf_{\text{hit}}$ newly retired hosts, and thus $ds(t) = x(t)f_{\text{hit}}f_{\text{old}}(t) + y(t)f_{\text{hit}}$.

From the above analysis, we have

$$f_{\text{hit}} = r \times dt \times \frac{V}{N} \tag{1}$$

$$f_{\text{old}}(t) = \frac{x(t) - \alpha(t)}{1 - i(t) + x(t) - \alpha(t)} \tag{2}$$

$$f_{\text{new}}(t) = \frac{1 - i(t)}{1 - i(t) + x(t) - \alpha(t)} = 1 - f_{\text{old}}(t) \tag{3}$$

$$f_{\text{ineff}}(t) = i(t) - (x(t) - \alpha(t)) \tag{4}$$

$$f_{\text{eff}}(t) = 1 - i(t) + x(t) - \alpha(t) = 1 - f_{\text{ineff}}(t) \tag{5}$$

$$di(t) = x(t)f_{\text{hit}}f_{\text{new}}(t) \tag{6}$$

$$dx(t) = x(t)f_{\text{hit}}f_{\text{new}}(t)f_{\text{eff}}(t) - x(t)f_{\text{hit}}f_{\text{old}}(t) \tag{7}$$

$$d\alpha(t) = x(t)f_{\text{hit}}f_{\text{new}}(t)f_{\text{eff}}(t) - \alpha(t)f_{\text{hit}} \tag{8}$$

$$dy(t) = x(t)f_{\text{hit}}f_{\text{new}}(t)f_{\text{ineff}}(t) - y(t)f_{\text{hit}} \tag{9}$$

$$ds(t) = x(t)f_{\text{hit}}f_{\text{old}}(t) + y(t)f_{\text{hit}} \tag{10}$$

$$da(t) = dx(t) + dy(t). \tag{11}$$

The boundary conditions to this set of equations are: $i(0) = a(0) = x(0) = \frac{v}{V}$, and $\alpha(0) = s(0) = y(0) = 0$, where v is the number of vulnerable hosts in the hitlist.

D. Verification of Our Model

We developed a simulator for permutation-scanning worms whose propagation strategy can be found in Section II-A and B. The simulator is implemented in C++ with proper encapsulation, i.e., a host object inside the simulator is not aware of the large picture of the network, and instead it can only see its own private variables, including its IP address, the state of its local random-number generator, the last address scanned, and the response to a scan message, i.e., new infection or not. Each host object h performs the following operations: If it is not infected, it does nothing. If it is infected, it uses a random number generator to produce a random location on the permutation ring and begins to sequentially scan the addresses along the ring after that location at a certain rate r. For each address to be scanned, host h first decrypts it through a decryption key that is shared by all infected hosts. The result is treated as an IP address, and h sends a message to the host object that owns the address. If that host is vulnerable but not yet infected, it becomes infected. However, if that host is already infected before receiving the message, it will inform h to retire. The controller object of the simulator performs the initial infection. At each time tick, it schedules the hosts in turn for their operations. The action of all hosts collectively simulates the propagation process of a permutation-scanning worm. The controller object records the numbers of infected, active, and retired hosts at the end of each time tick, from which we plot the worm propagation curves. Each vulnerable-host object uses the same decryption key, but has a different seed for the random number generator used for calculating the random location from which the host, after being infected, will begin its scanning. The simulation stops when all infected hosts retire.

Fig. 4 compares the propagation curves produced by the simulator to those generated by the analytical model for a 0-jump worm; the two sets of curves are nearly indistinguishable. The simulation parameters are given as follows. The size of the vulnerable population is $V = 2^{13}$. The hitlist contains $v = 100$ vulnerable hosts. The size of the address space is $N = 2^{23}$; it will take a prohibitively long time if N is chosen to be 2^{32}. To produce propagation curves in any of the figures in the paper, we simulate worm propagation for 1000 times under different random seeds, and then take the average. We normalize the time tick to be $\frac{1}{r}$. Namely, an infected host sends one scan message per tick. This allows the same propagation curves to be used for characterizing worm propagation under any scanning rate. Hosts with variable scanning rates will be investigated in Section VIII.

Worm propagation happens among end-hosts. It is not necessary to explicitly simulate the network topology. Because we are particularly interested in stealthy worms that scan at a low rate (e.g., one scan message every few seconds), we assume that the time tick—which is the inverse of scan rate—is larger than the Internet end-to-end delay (typically in tens or hundreds of milliseconds). In this case, infection will be completed within the current time tick, and the impact of the propagation delay of scan messages will be very small on the infection curve, which describes the percentage of vulnerable hosts that are infected over time. As we will further discuss in Section VIII, even when the Internet end-to-end delay is larger than the time tick, its impact on worm propagation is still small and quantifiable. We will also address other practical considerations, such as host patch and crash, network congestion, and bandwidth variability, in Section VIII.

E. Stealthiness of Permutation-Scanning Worms

The propagation curves in Fig. 4, which are computed from the model (1)–(11) or collected from the simulations, demonstrate the stealthiness of the permutation-scanning worms. The

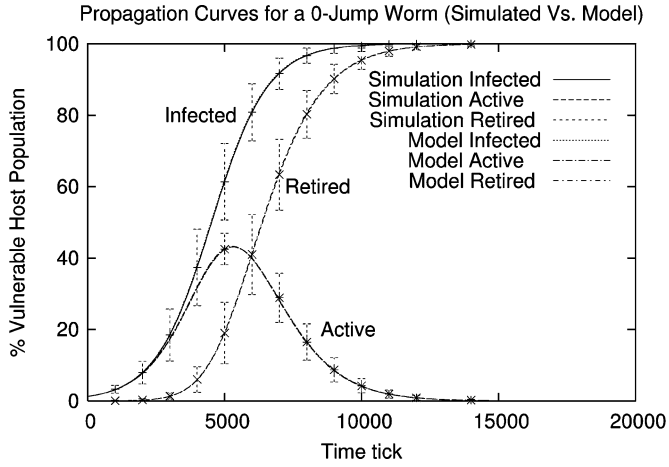

Fig. 4. The "Model" curves show the percentages of vulnerable hosts that are *infected*, *active*, and *retired* over time, respectively. These curves for $i(t)$, $a(t)$, and $s(t)$ are numerically computed from the analytical model in (1)–(11). The "Simulation" curves are plotted using the averaged data collected from the simulator; the 99% confidence intervals are also plotted for selected data points. As expected, the curves from the model and the curves from the simulator completely overlap, which verifies the correctness of the model.

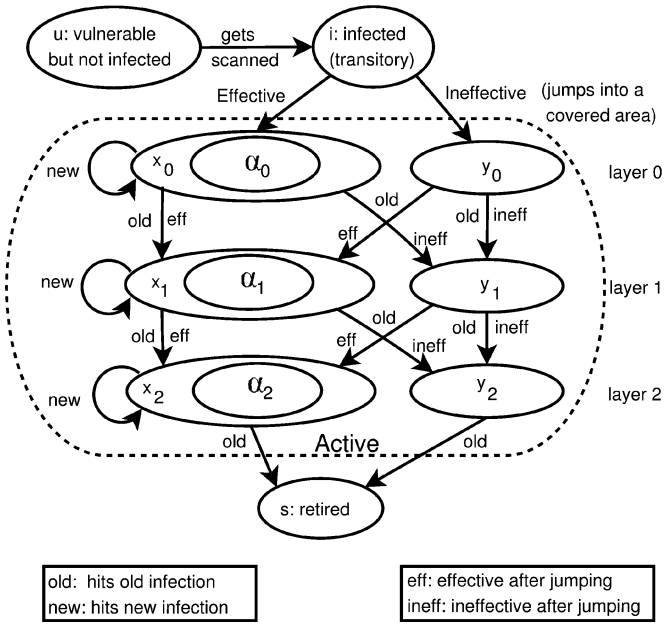

Fig. 5. State diagram of a k-jump worm with $k = 2$. In the diagram, we assign each active host a *layer number*, which indicates the number of old infections hit by the host. Once the host hits its $(k+1)$-th old infections, it retires immediately.

number of infected hosts, $Vi(t)$, grows over time and eventually becomes V. In the classical random-scanning worms, all infected hosts scan the Internet. The aggregate scan traffic peaks when all vulnerable hosts are infected. In the permutation-scanning worms, only the active hosts scan. The number of active hosts, $Va(t)$, can be much smaller than $Vi(t)$, which is evident from Fig. 4, where the active curve is below the infected curve. When the infected curve $i(t)$ peaks at 100%, the active curve $a(t)$ approaches to zero. That is, when all vulnerable hosts are infected, a random-scanning worm will reach the height of its scanning activity, whereas a permutation-scanning worm will entirely conceal its presence and stay stealthy. The total volume of scan traffic by a permutation-scanning worm, which corresponds to the area under the active curve, is bounded. The total scan volume by a random-scanning worm, which corresponds to the area under its infected curve, will be much larger because the area is open-ended (unless the infected hosts are scheduled to go dormant at certain times as Code Red does).

V. Extending the Model to k-Jump Worms

In this section, we demonstrate the flexibility of our analytical model by extending it for the k-jump worms. Modeling the propagation of k-jump worms is important as it will lead to a better understanding of the Warhol worm, which can infect the whole of Internet in a matter of minutes [12]. Warhol worms and permutation-scanning k-jump worms with large hitlists share similar propagation characteristics.

A. Further Classification of Active Hosts for k-Jump Worms

For a 0-jump worm, at any time t, none of the $a(t)$ active hosts has hit any old infection. However, for a k-jump worm, any active host (class x, α or y) could have hit anywhere between 0 to k old infections. While the terms $x(t)$, $\alpha(t)$, and $y(t)$ continue to denote the fractions of all vulnerable hosts that are effective (class x), nascent (class α), and ineffective (class y), respectively, at time t for a k-jump worm, each of those classes is further subdivided into $k+1$ subclasses depending on how many old infections they have already hit (between 0 and k). For example, class x is subdivided into classes $x_0, x_1, x_2 \cdots x_{k-1}, x_k$ such that $x(t) = \sum_{j=0}^{k} x_j(t)$, and similar notations are used for class α and y. For example, the total number of nascent hosts that have hit two old infections till time t are denoted by $\alpha_2(t)$.

Fig. 5 shows the state diagram that depicts how an active host moves from one subclass to another until it is retired. Each active host is assigned a *layer number*, which indicates the number of old infections hit by the host. Active hosts having already hit j old infections are referred to as j-layer hosts. When a j-layer host hits another old infection, it moves to layer $j+1$, or to the retired class if $j = k$.

We observe that the quantities, $f_{\text{old}}(t)$, $f_{\text{new}}(t)$, $f_{\text{eff}}(t)$, and $f_{\text{ineff}}(t)$ (defined in Section IV-B) stay the same for both 0-jump worms and k-jump worms. The analysis that produces the formulas for their calculation can be applied to both 0-jump worms and k-jump worms.

B. Interaction Among Scanning Hosts at Different Layers

The state transitions in Fig. 5 between subclasses at different layers are explained below.

- An active infected host never changes its layer by hitting a *new* infection. This is because the layer of a host indicates how many *old* infections the active host has hit till that time, and hitting a new infection does not change that. However, when it hits an old infection, it takes a jump, moves to the next layer, and becomes either ineffective or nascent depending on whether it jumps into a *covered area* or not. However, if it was already at the k-layer, then it retires after hitting its $(k+1)$-th old infection.

- Active hosts at *any* layer can hit a new infection. Therefore, when calculating change in $x_0(t), \alpha_0(t)$ and $y_0(t)$, we must consider the new infections caused by effective hosts at all $k+1$ layers.
- The number of active hosts at any layer, except for layer 0, will be changed only by the activity of the hosts at the same or previous layer. The number of hosts at a layer increases when hosts in the previous layer hit old infections and consequently move to this layer. Similarly, it decreases when hosts in this layer hit old infections and move to the next layer. Therefore, the derivative of the number of j-layer hosts, for $0 < j \leq k$, depends only on the numbers of hosts in the subclasses at layer j and layer $j-1$.

C. Propagation Model for k-Jump Worms

Below, we give the equations that model the propagation of k-jump worms. For the purpose of brevity, all symbols used in the formulas are function of time t, except for f_{hit}, V, and N, which are independent of time. For example, f_{new} denotes $f_{\text{new}}(t), d\alpha_j$ denotes $d\alpha_j(t)$, and so on. We omit the equations for $f_{\text{hit}}, f_{\text{old}}(t), f_{\text{new}}(t), f_{\text{eff}}(t)$, and $f_{\text{ineff}}(t)$ since they are the same as (1)–(5). The differential equations for k-jump worms are

$$dx_j = \begin{cases} \text{if } j = 0, & x f_{\text{hit}} f_{\text{new}} f_{\text{eff}} - x_j f_{\text{hit}} f_{\text{old}} \\ \text{if } 0 < j \leq k, & x_{j-1} f_{\text{hit}} f_{\text{old}} f_{\text{eff}} - x_j f_{\text{hit}} f_{\text{old}} \\ & \quad + y_{j-1} f_{\text{hit}} f_{\text{eff}} \end{cases} \tag{12}$$

$$d\alpha_j = \begin{cases} \text{if } j = 0, & x f_{\text{hit}} f_{\text{new}} f_{\text{eff}} - \alpha_j f_{\text{hit}} \\ \text{if } 0 < j \leq k, & x_{j-1} f_{\text{hit}} f_{\text{old}} f_{\text{eff}} - \alpha_j f_{\text{hit}} \\ & \quad + y_{j-1} f_{\text{hit}} f_{\text{eff}} \end{cases} \tag{13}$$

$$dy_j = \begin{cases} \text{if } j = 0, & x f_{\text{hit}} f_{\text{new}} f_{\text{ineff}} - y_j f_{\text{hit}} \\ \text{if } 0 < j \leq k, & x_{j-1} f_{\text{hit}} f_{\text{old}} f_{\text{ineff}} - y_j f_{\text{hit}} \\ & \quad + y_{j-1} f_{\text{hit}} f_{\text{ineff}} \end{cases} \tag{14}$$

$$dx = \sum_{j=0}^{k} dx_j \tag{15}$$

$$dy = \sum_{j=0}^{k} dy_j \tag{16}$$

$$d\alpha = \sum_{j=0}^{k} d\alpha_j \tag{17}$$

$$di = x f_{\text{hit}} f_{\text{new}} \tag{18}$$

$$da = dx + dy \tag{19}$$

$$ds = x_k f_{\text{hit}} f_{\text{old}} + y_k f_{\text{hit}}. \tag{20}$$

The boundary conditions at time $t = 0$ are $i(0) = a(0) = x(0) = x_0(0) = \frac{v}{V}$. All the other quantities $(s, x_1 \ldots x_k, \alpha, \alpha_0 \ldots \alpha_k, y, y_0 \ldots y_k)$ are zeros at $t = 0$.

D. Verification of the Correctness of the Model

For different values of k, we compare the result numerically computed from the model with the result collected from the simulator in Fig. 6, using the same experimental setup as described in Section IV-D. In all cases, the model and the simulation produce the same propagation curves.

VI. Closed-Form Solution for the 0-Jump Worm

In this section, we transform the set of equations in (1)–(11) to three simple differential equations that can be further integrated into the closed-form formulas for the numbers of infected, active, and retired hosts over time.

First, we establish a functional relation between $i(t)$ and $x(t)$. Recall that $i(t)$ is the fraction of the vulnerable host population that is infected at time t, and $x(t)$ is the fraction of the vulnerable host population that is actively scanning and can potentially generate new infections—more precisely, these so-called effective hosts are currently scanning addresses outside of any covered area. By definition, $i(t) = x(t) + y(t) + s(t)$. The infected hosts include effective hosts, ineffective hosts, and retired hosts.

We define a *current position* for each infected host. For an effective or ineffective host, its current position is the address it is scanning. For a retired host, its current position is the address it has scanned last before retirement. Interestingly, the current positions of all infected hosts are distributed along the permutation ring uniformly at random. That is because, right after infection, a host jumps to a location that is independently and randomly selected. As long as all infected hosts begin their scanning at independently random locations, their current positions will always be uncorrelated and statistically distributed along the ring uniformly at random.

By definition, the current position of an effective host will be outside of any covered area, and the current position of an ineffective or retired host will be in a covered area. Due to the random distribution of the current positions of all infected hosts, the fraction of infected hosts being effective is equal to the *fraction* of the permutation ring that is outside of the covered areas, *which* is simply $f_{\text{eff}}(t)$. From Section IV-D, we know that $f_{\text{eff}}(t) = 1 - f_{\text{ineff}}(t)$ and $f_{\text{ineff}}(t)$ equals the fraction of the ring that all covered areas together represent. Summarizing the above analysis, we have

$$x(t) = i(t) f_{\text{eff}}(t). \tag{21}$$

By plugging the above equation into (1)–(9), it can be easily verified that this equation is consistent with others in the model.

Applying (1), (3), (5), and (21) to (6), we have the following differential equation:

$$\frac{di(t)}{dt} = \frac{rV}{N} \times i(t) \times (1 - i(t)). \tag{22}$$

Applying (7) and (9) to (11), we have

$$da(t) = x(t) f_{\text{hit}} f_{\text{new}}(t) - x(t) f_{\text{hit}} f_{\text{old}}(t) - y(t) f_{\text{hit}}.$$

Because $y(t) = a(t) - x(t)$ by definition and $f_{\text{old}}(t) = 1 - f_{\text{new}}(t)$, we have

$$da(t) = 2x(t) f_{\text{hit}} f_{\text{new}}(t) - a(t) f_{\text{hit}}.$$

Applying (1), (3),(5) , and (21), we have

$$\frac{da(t)}{dt} = \frac{rV}{N} \times (2i(t) \times (1 - i(t)) - a(t)). \tag{23}$$

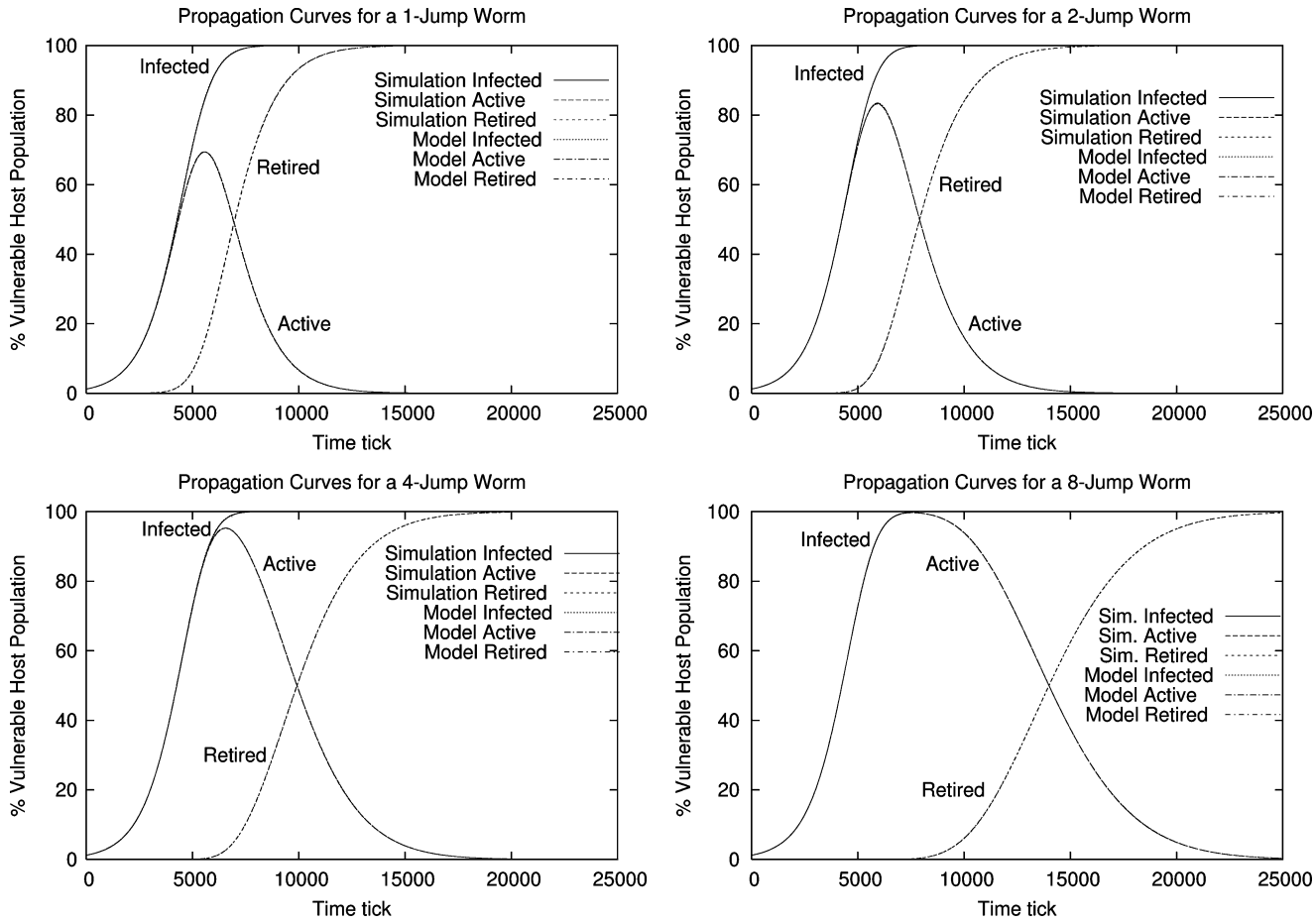

Fig. 6. The "Model" curves show the percentages of vulnerable hosts that are infected, active, and retired over time, respectively. These curves for $i(t)$, $a(t)$, and $s(t)$ are numerically computed from the analytical model in (12)–(20). The "Simulation" curves are plotted using the averaged data collected from the simulator. As expected, for $k = 1, 2, 4$, and 8, the curves from the model and the curves from the simulator completely overlap, which verifies the correctness of our model for k-jump worms.

Because $s(t) = i(t) - a(t)$ by definition, $\frac{ds(t)}{dt} = \frac{di(t)}{dt} - \frac{da(t)}{dt}$. From (22) and (23), we have

$$\frac{ds(t)}{dt} = \frac{rV}{N} \times (a(t) - i(t) \times (1 - i(t))). \tag{24}$$

Let $\phi = \frac{v}{V}$, which is the fraction of the vulnerable host population that is initially infected at time $t = 0$. Integrating (22)–(24), we have the following close-form solution:

$$i(t) = \frac{\phi e^{\frac{rV}{N}t}}{1 - \phi + \phi e^{\frac{rV}{N}t}} \tag{25}$$

$$a(t) = \frac{2(1-\phi)}{\phi e^{\frac{rV}{N}t}} \left\{ \frac{1-\phi}{1 - \phi + \phi e^{\frac{rV}{N}t}} - 1 + \phi + \ln\left(1 - \phi + \phi e^{\frac{rV}{N}t}\right) + \frac{\phi^2}{2(1-\phi)} \right\} \tag{26}$$

$$s(t) = \frac{\phi e^{\frac{rV}{N}t}}{1 - \phi + \phi e^{\frac{rV}{N}t}} - \frac{2(1-\phi)}{\phi e^{\frac{rV}{N}t}} \left\{ \frac{1-\phi}{1 - \phi + \phi e^{\frac{rV}{N}t}} - 1 + \phi + \ln\left(1 - \phi + \phi e^{\frac{rV}{N}t}\right) + \frac{\phi^2}{2(1-\phi)} \right\}. \tag{27}$$

VII. Usage of the Analytical Model

In this section, we first describe the benefits of having an analytical model when comparing with a simulator. We then analyze our model to see what impact each worm/network parameter (such as network size, vulnerable population size, hitlist size, and scanning rate) will have on the worm propagation.

A. Analytical Modeling or Simulation?

Properly simulating the worm propagation on the Internet at the packet level is very difficult due to its sheer scale. Even for a rather simplified version of the Internet, without an analytical model, one will need to take the average of multiple runs of a simulator in order to get acceptably reliable propagation curves. Since each run could potentially take a long time for realistic values of N and V, the whole process could take an enormous amount of time. For an imagined attack targeting at the Windows system, it took 16 h on an Intel Xeon 2.8-GHz processor with 4 GB RAM to run a *single* round of a simulation involving around 400 M potentially vulnerable windows hosts on IPv4 for one set of worm/network parameters. In order to run the same simulation for IPv6 ($N = 2^{128}$), it is easy to see that the runtime will be astronomical. On the contrary, a *single* run of the numerical simulation based on the analytical model takes just seconds and gives us the provably correct results. Moreover, it can

TABLE I
TIME (IN SECONDS) IT TAKES TO INFECT 50% OF ALL VULNERABLE HOSTS, WHERE $\alpha = 50\%, N = 2^{32}, V = 1\,000\,000, r = 1$ PER SECOND, AND ϕ VARIES FROM 0.00001 TO 0.05

ϕ	0.00001	0.0001	0.001	0.01	0.05
t_α	49448	39558	29664	19736	12646

TABLE II
TIME (IN SECONDS) IT TAKES TO INFECT 50% OF ALL VULNERABLE HOSTS, WHERE $\alpha = 50\%, \phi = 0.0001, N = 2^{32}, V = 1\,000\,000$, AND r VARIES FROM 100 TO 0.01 PER SECOND

r	100	10	1	0.1	0.01
t_α	396	3956	39558	395577	3955768

handle extremely large address spaces and vulnerable host populations. For any worm/network parameter change, new propagation curves can be recomputed in little time for comparison.

While arguments can be made for doing a scaled-down simulation and then scaling up the results, such simulations are often not fully accurate and suffer from stochastic fluctuations and other problems [15]. Moreover, such simulations cannot predict with confidence what *precise* effect each worm/network parameter will have on the overall outcome and for what reason. In comparison, an analytical model can tell *exactly* why and by how much a parameter will affect the outcome.

B. How Will the Worm/Network Parameters Affect a Worm's Propagation?

Below, we analyze the exact effect of each worm/network parameter on the worm propagation.

- **Effect of Address Space Size** (N): For either 0-jump worms or k-jump worms, the only term in the model that is directly affected by N is $f_{\text{hit}} = r \times dt \times \frac{V}{N}$ in (1). Since all the incremental terms [such as $di(t), da(t)$, and $ds(t)$] are direct multiples of f_{hit}, the first derivatives of the propagation curves for infected, active, and retired hosts are inversely proportional to N. The first derivative characterizes the rate of change over time. Therefore, as the size of the address space increases, a worm propagates inverse-proportionally slower. If the address space is doubled, it will take the worm double the amount of time to infect all vulnerable hosts. This gives a reason for transition to IPv6.
- **Effect of Vulnerable Host Population Size** (V): For either 0-jump worms or k-jump worms, the only term in the model that is affected by V is $f_{\text{hit}} = r \times dt \times \frac{V}{N}$. Again, because the incremental terms [such as $di(t), da(t)$, and $ds(t)$] are direct multiples of f_{hit}, the first derivatives of the propagation curves for infected, active, and retired hosts are proportional to V. As V increases, a worm propagates proportionally faster. If V is doubled, it takes the worm half the amount of time to infect all vulnerable hosts.
- **Effect of Hitlist Size** (v): The size of the hitlist is controlled by the worm designer. As per our observations from the analytical model, a higher v simply shifts the infection curve, $i(t)$, to the left on the time axis with $i(0) = \frac{v}{V}$. From the infection curve in Fig. 4, we see that there is a slow start phase where $i(t)$ increases slowly before it transitions into a rapid growth phase. A larger hitlist will shorten the initial slow start phase and reduce the overall propagation time. More specifically, from (25), the time it takes to infect a percentage α of all vulnerable hosts is

$$t_\alpha(\phi) = \frac{N}{r \cdot V}\left(\ln\left(\frac{1}{\phi} - 1\right) - \ln\left(\frac{1}{\alpha} - 1\right)\right) \qquad (28)$$

which is a decreasing function with respect to $\phi = \frac{v}{V}$. Table I shows the numerical results computed from (28). It demonstrates that the worm's propagation time can be significantly reduced by increasing the hitlist size.

- **Effect of Scanning Rate** (r): Again, for either 0-jump worms or k-jump worms, the only term in the model that is affected by r is $f_{\text{hit}} = r \times dt \times \frac{V}{N}$. Since the incremental terms [such as $di(t), da(t)$, and $ds(t)$] are direct multiples of f_{hit}, the first derivatives of the propagation curves for infected, active, and retired hosts are inversely proportional to r. Thus, if the scanning rate is doubled, the time it takes a worm to infect the vulnerable host population will be halved.
  There exists a tradeoff between a worm's stealthiness and its infection speed. When r is smaller, the stealthiness of the worm is improved; it is certainly easier to identify an aggressive infected host that scans 100 different addresses per second than a host that scans one address per minute, making itself look like a normal one. However, when r is smaller, it will take longer for the worm to infect a certain percentage of all vulnerable hosts. In (28), if we treat ϕ as a constant and r as a variable, then t_α becomes a function of r. Table II shows the numerical results computed from (28), which demonstrates the tradeoff: As r decreases, t_α increases inversely proportionally.
- **Effect of Varying k for a k-jump Worm**: We make an important observation from Fig. 7, where the infection speed and the scanning volume of a k-jump worm are presented for different k values. The *scanning volume* is defined as the total scanning traffic generated by all active hosts since time $t = 0$. We see that by increasing k, the slope of the infection curve in the left plot is somewhat steeper, but for $k > 1$, the incremental gain becomes negligible. On the other hand, with a higher k, the onset of retirement for active hosts happens at an increasingly later time, which means larger scanning volume, as shown in the right plot. We observe that, for $k \geq 8$, almost *all* infected hosts are active at the time when all vulnerable hosts have been infected, producing a big network footprint for worm detection. Therefore, it makes little sense to deploy a k-jump worm with a high value of k.

VIII. PRACTICAL CONSIDERATION

In this section, we consider our model under real-world considerations, including congestion and bandwidth variability, patching and host crash, as well as delay of scan messages.

A. Congestion and Bandwidth Variability

If for stealth reasons the worm sets a small scanning rate r such as 100 per minute, most infected hosts are likely to have the bandwidth of delivering 100 SYN packets per minute, and our

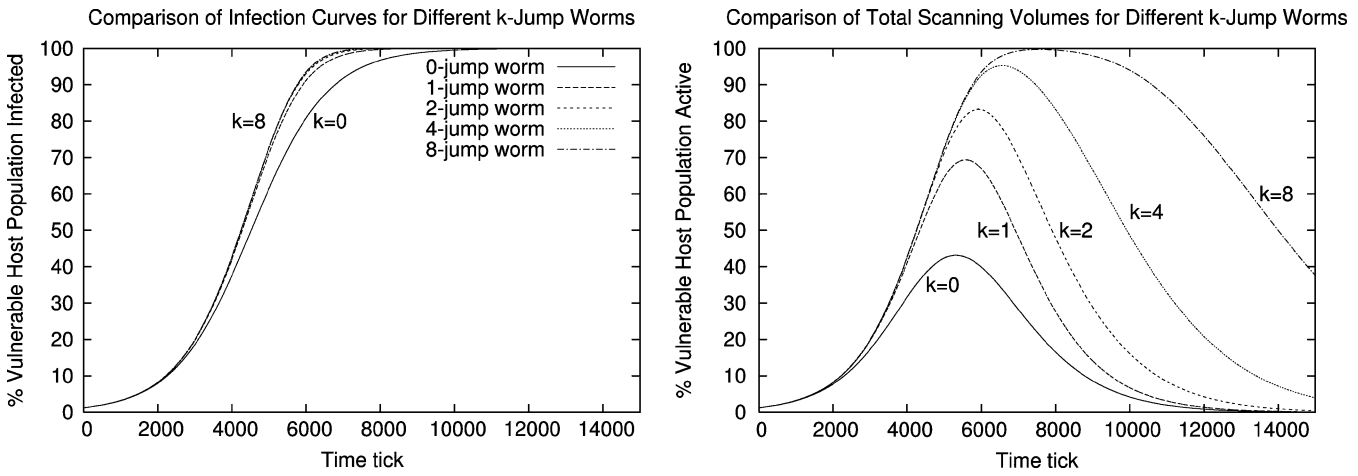

Fig. 7. The left plot shows the infection curves, $i(t)$, for a k-jump worm under different k values. The right plot shows the active curves, $a(t)$, for a k-jump worm under different k values. Recall that $a(t)$ is the percentage of vulnerable hosts that are actively scanning at time t. The total amount of scanning traffic, called the *scanning volume*, is defined as the area under the active curve. From the left plot, we see that the infection speed improves when k increases. However, the rate of improvement diminishes quickly when k is greater than 1. On the other hand, from the right plot, the scanning volume increases *significantly* when k increases.

model will be accurate if the deviation caused by Internet delay is negligible. However, if the worm sets its scanning rate r to be 10 000 per second, then the actual scanning rates of infected hosts may vary due to network congestion. We believe a worm that causes network congestion is not a good worm because it loses stealth (unless its sole purpose is to create headlines by service disruption, which is rarely the case nowadays [17]).

Congestion also happens naturally in the network without worm activity due to the bandwidth limitation and the demand on the routers. As long as the Internet is able to deliver the low scanning rate of most infected hosts, our model can predict the propagation behavior of low-rate stealthy worms. However, we realize that whatever be the reason—processing power of infected host, available bandwidth for the user, congestion of the network—the final result is that on the Internet scale, different hosts are in effect scanning at different rates. Therefore, if we can somehow extend our model to accommodate variable scanning rates from different hosts, we are effectively capturing the real network situation arising out of the reasons mentioned above. Since our model can handle only a fixed scanning rate, we posited that by using average scanning rate, our model should be able to still approximate the variable scanning rate scenario. With that goal in mind, we simulated two worms, one having a fixed rate, $r = 5$ per time tick for all infected hosts, and the other having variable rates with the Gaussian distribution and a mean value of 5 per time tick. Fig. 8 shows that the infection curves of the two worms are very close. Similar results are observed for other variable rate distributions. Thus, the model is able to approximate the propagation of worms by using average scanning rate. Therefore, we argue that our model is indeed able to approximate the propagation of worms in real-life scenarios by using the average scanning rate.

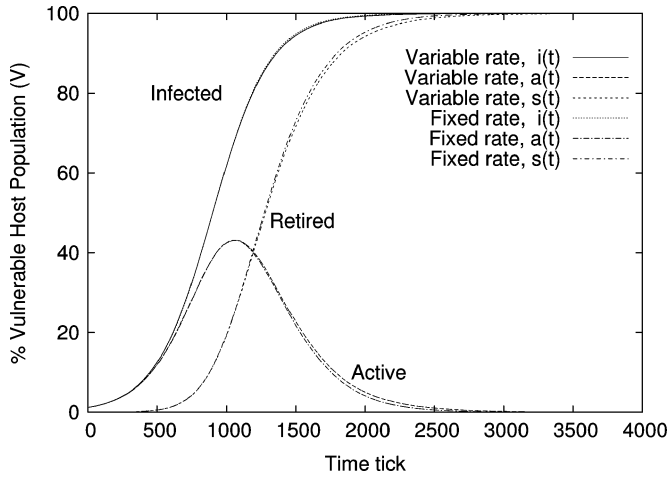

Fig. 8. Comparison of propagation curves for worms with variable- and fixed-rate of scanning. The simulation parameters used are $N = 2^{23}, V = 2^{13}$, and $v = 100$. The value of r is a constant five scans per time tick for the fixed-rate scanning worm, and the other having variable rates with the Gaussian distribution and a mean value of 5 per time tick.

B. Patching and Host Crash

Once a vulnerable host is infected and starts scanning, it may be removed from the vulnerable host population due to multiple reasons. First, upon infection the host may simply crash. Second, the host may be patched by the system administrator after some time. Third, due to scanning activity, an infected host may come under suspicion of the network administrator and resultingly taken off the network or quarantined. We show that our model can be extended to handle the removal of vulnerable hosts.

We introduce a few additional terms in our model to account for the removal of hosts. Let p_q denote the probability of a host being removed each time it scans, and $q(t)$ denote the number of vulnerable hosts that are removed from the system by time t. As hosts are removed, the vulnerable population also changes; we use $V(t)$ for the number of vulnerable hosts at time t. It is evident that $V(t) + q(t) = V(0)$ for all t. However, under this "removal" scheme the meaning of $i(t)$ becomes unclear as some hosts that were infected can later be disinfected. To clear this confusion, we introduce a third new term called $i_{\text{ever}}(t)$ to denote the fraction of *original* vulnerable host population $(V(0))$ that were *ever* infected during the whole propagation, while $i(t)$

denotes the fraction of $V(t)$ that are currently infected. Since $V(t)$ is not a constant, we rather plot $i_{\text{ever}}(t)$.

With these new terms, we rewrite the propagation equations of a 0-jump worm by considering host removal

$$\begin{aligned}
f_{\text{hit}}(t) &= r \times dt \times \frac{V(t)}{N} \\
f_{\text{old}}(t) &= \frac{x(t)-\alpha(t)}{1-i(t)+x(t)-\alpha(t)} \\
f_{\text{new}}(t) &= \frac{1-i(t)}{1-i(t)+x(t)-\alpha(t)} = 1 - f_{\text{old}}(t) \\
f_{\text{ineff}}(t) &= i(t) - (x(t)-\alpha(t)) \\
f_{\text{eff}}(t) &= 1 - i(t) + x(t) - \alpha(t) = 1 - f_{\text{ineff}}(t) \\
dx(t) &= x(t) f_{\text{hit}}(t) f_{\text{new}}(t) f_{\text{eff}}(t) - x(t) f_{\text{hit}}(t) f_{\text{old}}(t) \\
&\quad - x(t) p_q \\
d\alpha(t) &= x(t) f_{\text{hit}}(t) f_{\text{new}}(t) f_{\text{eff}}(t) - \alpha(t) f_{\text{hit}}(t) \\
&\quad - \alpha(t) p_q \\
dy(t) &= x(t) f_{\text{hit}}(t) f_{\text{new}}(t) f_{\text{ineff}}(t) - y(t) f_{\text{hit}}(t) \\
&\quad - y(t) p_q \\
ds(t) &= x(t) f_{\text{hit}}(t) f_{\text{old}}(t) + y(t) f_{\text{hit}}(t) \\
da(t) &= dx(t) + dy(t) \\
dq(t) &= x(t) p_q + y(t) p_q \\
di(t) &= x(t) f_{\text{hit}}(t) f_{\text{new}}(t) - dq(t) \\
di_{\text{ever}}(t) &= di(t) + dq(t) \\
dV(t) &= -dq(t)
\end{aligned}$$

The boundary conditions to this set of equations are $i(0) = a(0) = x(0) = \frac{v}{V(0)}$, and $\alpha(0) = s(0) = y(0) = 0$.

C. Internet Delay

When deriving the propagation model, we implicitly assume that each scan message instantaneously reaches the address being scanned. In reality, the worm will propagate slower due to end-to-end delay of the Internet. Hence, the model in (25) gives an upper bound on the worm's propagation speed.

In case of a new infection using TCP, it takes one round trip to exchange SYN (which is the scan message) and SYN/ACK, and then it takes a number of round trips to transmit ACK and attack packets. For example, if the worm code is 3 kB long and each TCP segment is 512 bytes, then under TCP's slow start, it takes three round trips to complete the infection. Internet's round-trip delay rarely exceeds 1 s [18]. Let D be a time period that upper-bounds the delay of most infections. Since worm code is typically short (in order to fit in the call stack without causing the program to crash when buffer-overflow attack is used), D is expected to be no more than several seconds.

The larger the infection delay is, the slower the worm propagates. Hence, if we artificially set the delay of all infections to the upper bound D (ignoring the rare cases whose delay exceeds D), we have a lower bound on the worm propagation speed. It can be shown that this lower bound is simply the infection curve (25) shifted to the left by D. Combining both the lower bound and the upper bound, we have the following inequality for the actual value of $i(t)$ after Internet delay is considered. For $t \geq D$

$$\frac{\phi e^{\frac{rV}{N}(t-D)}}{1-\phi+\phi e^{\frac{rV}{N}(t-D)}} \leq i(t) \leq \frac{\phi e^{\frac{rV}{N}t}}{1-\phi+\phi e^{\frac{rV}{N}t}}. \tag{29}$$

If a worm wants to stay undetected, it will choose a low scanning rate for better stealthiness (smaller footprint on the Internet) even when that means lower propagation speed and longer propagation time. Many known worms take hours or tens of minutes to infect the Internet. For these worms, a maximum deviation of several seconds by the model from the reality is relatively small with respect to the much longer overall propagation time. Note that our goal here is not to determine the actual value of D. Instead, we argue that the predictive power of our model is relevant in reality when the Internet delay is small comparing with the worm propagation time.

IX. Conclusion

In this paper, we have successfully modeled the propagation characteristics of different varieties of permutation-scanning worms. We first derive the precise propagation model for 0-jump worms, and then extend it for the general k-jump worms. We are also able to derive the closed-form solution for the 0-jump worms. To verify the correctness of the model, we compare the results from our model with those obtained from actual worm simulations and show that they perfectly match. We analyze the model to demonstrate how each worm/network parameter will affect the worm's propagation behavior. Finally, although our analytical model was originally conceived by assuming ideal network conditions, we show that it can very well be extended to real-life scenarios with the consideration of variable host bandwidth, network congestion, Internet delay, host crash, and patching. In our future work, we will continue refining our model by considering other practical extensions. One interesting direction is to study how local subnet scanning that has been employed in some existing worms may be incorporated into the permutation-scanning worms and how that will affect the propagation model. Another direction is to directly include variable scanning rates into the model's equations, instead of using the average scanning rate as approximation when variable rates exist (Section VIII-A).

References

[1] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the slammer worm," in *Proc. IEEE Security Privacy*, Jul. 2003, vol. 1, no. 4, pp. 33–39.

[2] Z. Chen and C. Ji, "Optimal worm-scanning method using vulnerable-host distributions," *Int. J. Security Netw.*, vol. 2, no. 1/2, pp. 71–80, 2007, Special Issue on Computer and Network Security.

[3] M. Vojnovic, V. Gupta, T. Karagiannis, and C. Gkantsidis, "Sampling strategies for epidemic-style information dissemination," in *Proc. IEEE INFOCOM*, Apr. 2008, pp. 1678–1686.

[4] J. Ma, G. M. Voelker, and S. Savage, "Self-stopping worms," in *Proc. ACM Workshop Rapid Malcode (WORM)*, 2005, pp. 12–21.

[5] "Symantec Internet security threat report," Apr. 2008 [Online]. Available: http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepape r_internet_security_threat_report_xiii_04-2008.en-us.pdf

[6] S. Chen and Y. Tang, "Slowing down Internet worms," in *Proc. 24th ICDCS*, Mar. 2004, pp. 312–319.
[7] G. Gu, M. Sharif, X. Qin, D. Dagon, W. Lee, and G. Riley, "Worm detection, early warning and response based on local victim information," in *Proc. 20th ACSAC*, 2004, pp. 136–145.
[8] S. Schechter, J. Jung, and A. W. Berger, "Fast detection of scanning worm infections," in *Proc. 7th Int. Symp. Recent Adv. Intrusion Detection*, Sep. 2004, pp. 59–81.
[9] Z. Li, M. Sanghi, Y. Chen, M. Kao, and B. Chavez, "Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience," in *Proc. IEEE Symp. Security Privacy*, 2006, pp. 33–47.
[10] Z. Chen and C. Ji, "Measuring network-aware worm spreading ability," in *Proc. IEEE INFOCOM*, May 2007, pp. 116–124.
[11] Z. Li, L. Wang, Y. Chen, and Z. Fu, "Network-based and attack-resilient length signature generation for zero-day polymorphic worms," in *Proc. IEEE ICNP*, 2007, pp. 164–173.
[12] S. Staniford, V. Paxson, and N. Weaver, "How to own the Internet in your spare time," in *Proc. 11th USENIX Security Symp.*, Aug. 2002, pp. 149–167.
[13] C. C. Zou, W. Gong, and D. Towsley, "Code red worm propagation modeling and analysis," in *Proc. 9th ACM Conf. Comput. Commun. Security*, Nov. 2002, pp. 138–147.
[14] Z. Chen, L. Gao, and K. Kwiat, "Modeling the spread of active worms," in *Proc. IEEE INFOCOM*, Mar. 2003, vol. 3, pp. 1890–1900.
[15] N. Weaver, I. Hamadeh, G. Kesidis, and V. Paxson, "Preliminary results using scale-down to explore worm dynamics," in *Proc. ACM Workshop Rapid Malcode (WORM)*, Mar. 2004, pp. 65–72.
[16] G. Yan and S. Eidenbenz, "Modeling propagation dynamics of Bluetooth worms," in *Proc. ICDCS*, 2007, p. 42.
[17] E. E. Schultz, "Where have the worms and viruses gone? New trends in malware," *Comput. Fraud Security*, vol. 2006, no. 7, pp. 4–8, Aug. 2006.
[18] A. Corlett, D. I. Pullin, and S. Sargood, "Statistics of one-way Internet packet delays," Mar. 2002 [Online]. Available: http://www3.ietf.org/proceedings/02mar/slides/ippm-4.pdf

Parbati Kumar Manna (M'09) received the B.Tech. degree from the Indian Institute of Technology, Kharagpur, India, in 1997, and the M.S. and Ph.D. degrees in computer and information science and engineering from the University of Florida, Gainesville, in 2007 and 2008, respectively.

Between 1997 and 2002, he worked in the renowned Indian software company Infosys Technologies Ltd. After obtaining his Ph.D. degree, he joined the Security Center of Excellence (SeCoE) at Intel, Hillsboro, OR. He held prestigious National Talent Search Examination (NTSE) and Merit scholarships endowed by the Government of India. His research area includes malware propagation and detection, designing malware of the future, and intrusion detection.

Shigang Chen received the B.S. degree in computer science from the University of Science and Technology of China, Hefei, China, in 1993, and the M.S. and Ph.D. degrees in computer science from the University of Illinois at Urbana-Champaign in 1996 and 1999, respectively.

After graduation, he worked with Cisco Systems, San Jose, CA, for three years before joining the University of Florida, Gainesville, in 2002, where he is currently an Associate Professor. His research interests include network security and wireless networks.

Dr. Chen received the IEEE Communications Society Best Tutorial Paper Award in 1999 and the NSF CAREER Award in 2007. He was a Guest Editor for the *ACM/Baltzer Journal of Wireless Networks* (WINET) and the IEEE TRANSACTIONS ON VEHICLE TECHNOLOGIES. He served as a Technical Program Committee (TPC) Co-Chair for the Computer and Network Security Symposium of IEEE IWCCC 2006, a Vice TPC Chair for IEEE MASS 2005, a Vice General Chair for QShine 2005, a TPC Co-Chair for QShine 2004, and a TPC member for many conferences including IEEE ICNP, IEEE INFOCOM, IEEE ICC, IEEE Globecom, etc.

Sanjay Ranka (F'02) received the B.Tech. degree in computer science from the Indian Institute of Technology, Kanpur, India, in 1985, and the Ph.D. degree in computer science from the University of Minnesota, Minneapolis, in 1988.

He is a Professor with the Department of Computer Science, University of Florida, Gainesville. He is a world-renowned technologist in the areas of large-scale software systems, high-performance networking, interactive marketing, CRM, data mining, and optimization. Most recently, he was the Chief Technology Officer at Paramark, Sunnyvale, CA, where he developed real-time optimization software for optimizing marketing campaigns. He has also held positions as a tenured faculty member at Syracuse University, Syracuse, NY, and as a Researcher/Visitor at IBM T.J. Watson Research Labs, Hawthorne, NY, and Hitachi America Limited, Sunnyvale, CA. He has co-authored two books: *Elements of Neural Networks* (Cambridge, MA: MIT Press, 1996) and *Hypercube Algorithms* (New York: Springer-Verlag, 1990).

Prof. Ranka is a Fellow of the AAAS and a Member of the IFIP Committee on System Modeling and Optimization.