Cooperative Containment of Fast Scanning Worms

Jayanthkumar Kannan, Lakshminarayanan Subramanian,
Ion Stoica, Scott Shenker, Randy Katz

Report No. UCB/CSD-04-1359

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Cooperative Containment of Fast Scanning Worms

Jayanthkumar Kannan, Lakshminarayanan Subramanian,
Ion Stoica, Scott Shenker, Randy Katz

Abstract

Scanning worms, that spread by probing the IP address space to find vulnerable hosts, are among the most serious threats to Internet security today, as evident by the time-scales of some recent large-scale worm attacks. Only an automatic defense can hope to contain a carefully designed worm that uses an unknown or a recently-divulged vulnerability. In this paper, we propose a *cooperation-based worm containment* approach that enables potentially distrusting firewalls in different access networks to exchange information in order to *contain* the spread of a fast scanning worm. Based on modeling the propagation of scanning worms, we identify and analytically quantify the effectiveness of two forms of cooperation between firewalls, namely, *implicit signaling* and *explicit signaling*. Specifically, we highlight the regimes under which implicit and explicit signaling provide effective containment. In this paper, we also address some of the *deployment* challenges associated with cooperation-based worm containment. Specifically, in a partial deployment scenario where only a small fraction of access networks (1%) are protected behind firewalls, we demonstrate a *rerouting* mechanism that can provide effective containment (97%) for these protected networks. One limitation of our work is that our analysis does not apply to worms based on pre-generated target lists, stealthy worms that slowly infect their vulnerable population, and rapidly mutating polymorphic worms.

1 Introduction

Scanning worms that probe addresses in order to find vulnerable hosts is the most common class of worms today. Several recent worms, such as Slammer [1], Witty [2], CodeRed [3, 4], Blaster [5], Nimda [6], fall in this category. Carefully designed and programmed scanning worms [7] that can infect most of the vulnerable population in a matter of minutes have been predicted in theory and observed in practice. Slammer, one of the fastest scanning worms seen so far, took only 10 minutes to infect 90% of the Internet's vulnerable population [1]. Moreover, the possibility of zero-day worm attacks based on an unknown vulnerability has also been suggested [8]. On the other hand, many existing worm defenses like patching, address blacklisting hold out little hope in containing fast-spreading worms [9].

In this work, we explore cooperation as a paradigm for containing fast-scanning worms. By cooperation, we refer to the exchange of information among firewalls in different access networks to collectively detect and contain worms. Cooperation can potentially have much better containment than standalone mechanisms since firewalls of infected networks can signal the spread of the worm to other firewalls in uninfected networks who can then install worm-filters to escape infection.

Cooperation however comes with its own challenges: incremental deployability, scalability, and security. Any solution for cooperation should support *incremental deployment* and should provide effective containment even when all firewalls in the Internet do not participate. While cooperation can potentially perform better if more firewalls participate, one challenge is to ensure that the information exchange can *scale* with the number of firewalls. Our scheme should also be able to accommodate firewalls that willfully or accidentally propagate incorrect information. A cooperative scheme should be *resilient to false alarms* triggered by such erroneous information.

The notion of cooperation has been suggested earlier in literature. There have been several proposed detection and containment architectures that involve participation among multiple firewalls (*e.g.*, Hard Perimeters [10], Domino [11], Wormholes [12], Weaver et al [13]). However, some of them assume a complete deployment scenario, and most of them assume that the participants trust one another. Cooperation between mutually untrusted participants has only recently been explored (in [14], [15]): the efficacy of these mechanisms is not fully understood and moreover, their performance degrades with the scanning rate of the worm. Our work generalizes on such existing work and aims to place the cooperation paradigm on a sound theoretical footing. Such an analysis helps to identify the effectiveness and limitations of this approach. Our solution is also fully decentralized and this precludes targeted attacks against a few participants in the system from affecting the performance of the entire system.

In our cooperative containment scheme, a firewall can alert other firewalls of an ongoing infection by using two forms of information propagation: *implicit* and *explicit* signaling. Implicit signaling marks suspect malicious packets in order to send alerts to other firewalls, while explicit signaling involves an exchange of alerts between firewalls. These mechanisms enable firewalls that have *detected* infection to propagate signals to other firewalls which can then *filter* their incoming traffic based on these signals. In order to improve the containment when only some Internet firewalls follow our protocol, we propose a technique called *rerouting* where traffic from undeployed firewalls is also monitored. We finally discuss attacks by malicious firewalls and smart scanning worms against our scheme and extend the rerouting mechanism to deal with malicious participants. Though we have only considered scanning worms, using suitable local detection schemes, our results on propagation may also *generalize* to other classes of worms [8]. Our cooperative schemes do not sacrifice *privacy* since they only divulge the source and destination addresses of traffic seen by the firewall: in particular,

packet contents are not revealed.

The primary metric we use to evaluate the effectiveness of our scheme is the *containment metric* which represents the fraction of vulnerable networks that escaped infection. We used analytical techniques and numerical analysis to derive the following results:

- **Complete Deployment:** We derive analytical bounds on the containment offered by our schemes under complete deployment. We show that if vulnerable hosts are sparsely distributed in the address space, then local detection and filtering (without any propagation) is sufficient to contain a scanning worm. For the fastest scanning worms known today, implicit signaling can achieve 97% containment and explicit signaling achieves 99.8% containment.
- **Partial deployment**: Our results indicate that even if only 1% of networks are behind deployed firewalls, our scheme can protect over 92% of such networks against the fastest scanning worms known today (and over 97% with rerouting, which however does not work if the worm can spoof source addresses).
- **Scalability**: Implicit signaling requires minimal communication overhead, but its containment degrades logarithmically with the number of participating firewalls N. Explicit signaling, on the other hand, can achieve a constant containment factor independent of N at the expense of $log(N)$ communication overhead per firewall.
- **Security**: Our scheme is robust to false alarms triggered by a few hundred malicious or unreliable firewalls in an Internet-scale cooperative. However, our scheme cannot handle the case in which a substantial fraction of firewalls might be malicious (*e.g.,* the worm subverts firewalls).
- **Tradeoffs between implicit and explicit signaling:** Implicit signaling has low traffic overhead and its performance is independent of the scanning rate of the worm, while explicit signaling is more resilient to malicious firewalls and smart scanning worms at the expense of more overhead.

Limitations: Our work only deals with fast spreading scanning worms that require automatic response. Stealthy worms that propagate slowly can fly under the radar of our mechanisms: such worms require other means of detection and control and are beyond the scope of this work. We do not address other forms of worms such as hit-list based worms [8]. In our work, we have assumed that, given suitable samples of worm traffic, a firewall can identify malicious packets. This identification can be based on a simple byte signature or other methods such as Autograph [16].

We have obtained results using a combination of analysis and simulation based on a simple model of the Internet. While we have tried to model the primary factors characterizing a worm attack, a full-scale implementation would be required to fully validate our results. Our scheme uses random sampling techniques to defend against malicious firewalls and can handle up to a few hundred such firewalls in an Internet-scale cooperative. Auditing mechanisms can be used to discourage large scale malicious behavior.

Outline: The rest of the paper is structured as follows. We provide a precise problem definition in Section 2. We then describe our solution under simplifying assumptions in Section 3, and analyze its effectiveness in Section 4. We consider the incremental deployment requirement in Section 5, and address security issues in Section 6. We discuss related work in Section 7 and present our conclusions in Section 8.

2 Problem Definition

The problem that we address in this work is stated as follows. *Consider N mutually distrusting firewalls on the Internet, each of which monitors an access network and desires to defend its network against a scanning worm. How should these firewalls cooperate with one another so as to minimize the containment metric?* In this section, we qualify this problem description further by specifying our simplified model of the Internet and the class of worms we wish to defend against. We then discuss the primary evaluation metric and the desirable features of a cooperation-based scheme.

Network Model: In our network model, we consider the Internet to consist of N access networks that are connected to the Internet core through an access link to their ISPs. We further assume that these access links are monitored by firewalls and that a host does not belong to more than one access network. Thus, any traffic exchanged between hosts in two different access networks can be monitored by the firewalls of these networks. This implies there are no covert links between hosts in different networks. Of course, traffic exchanged between hosts in the same access network will not be seen by their firewall. This implies that if a *single* host in a network is infected, its firewall cannot prevent other hosts within the same network from being infected. We expect that there would be a few hundred thousand firewalls in our cooperative (based on the number of observed BGP prefixes [17]).

Worm Model: The class of worms that we focus on are fast scanning worms that find vulnerable hosts by scanning the IP address space. We assume that the scanning pattern of a worm is as follows: an infected host first infects all hosts in its network in zero time and then begins probing external IP addresses uniformly at random to find more vulnerable hosts. This corresponds to a local topological scan followed by a global uniform random scan. Later, we will extend this model in order to analyze non-uniform random scanning worms. We do not consider worms that find vulnerable hosts by means such as consulting search engines, global host directories, pre-generated hit lists [7].

Goals: Our primary metric of success against a worm attack is the fraction of vulnerable networks that escape infection. We refer to this as the *containment* metric. We count uninfected networks instead of hosts, since eventually, either all vulnerable hosts belonging to the same network are infected or none of them are infected. We now discuss the desirable features of a cooperative containment scheme: deployability, secure cooperation, and scalability. First, due to the scale of

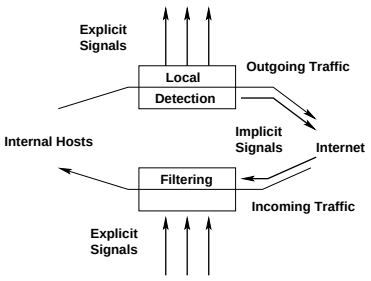

Figure 1: Firewall Activities

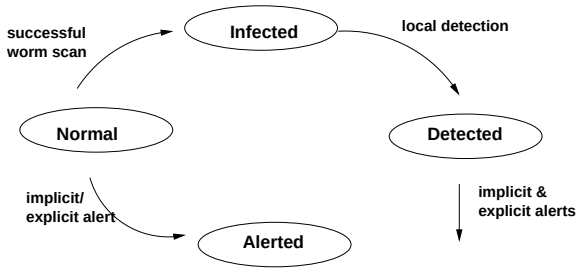

Figure 2: State Transition

the Internet, even if only some of the firewalls choose to implement our scheme, they should be able to reap the benefits of the scheme. We refer to this as the *partial deployment* scenario. Second, the effectiveness of a cooperative scheme would improve as more and more firewalls in the Internet followed our scheme. However, this means that the participants cannot trust one another: some firewalls could be malicious, or could be simply unreliable. Thus, our scheme should be *robust* to both false negatives (missing a worm attack) and false positives (raising a false alarm). In particular, *false positives* is one of the pitfalls of automatic response systems, and our scheme should be robust to malicious behavior attempting to trigger such false positives. Finally, the overhead of cooperation should *scale* with the number of participating firewalls.

3 Idealized Model

In this section, we propose a solution for worm containment that works in an idealized setting, where we make the following two assumptions (which will be removed later):

1. Complete Deployment: Every firewall in the Internet participates in our cooperative.
2. Trustworthy firewalls: All firewalls are trustworthy.

These assumptions considerably simplify the solution since one does not have to deal with adversaries who try to attack our cooperative by triggering false alarms. We first describe the high-level components of our solution, and then describe each component in detail.

3.1 Solution Framework

Firewalls participating in our cooperative perform the following three functions (illustrated in Figure 1).

Local Detection: The goal of local detection is for a firewall to determine whether its own network is infected. A firewall monitors its outgoing traffic to perform local detection. There are several advantages to analyzing outgoing traffic as opposed to analyzing incoming traffic. A firewall can maintain characteristics of its outgoing traffic using per-host state, unlike incoming traffic which can be very noisy due to routine Internet crud. Moreover, a decision made using incoming traffic can potentially be influenced by external malicious hosts sending traffic to the firewall's network. Another advantage of analyzing outgoing traffic is that malicious packets can be dropped at the source firewall before entering the Internet. However analyzing outgoing traffic cannot aid a firewall in protecting its *own* network, since the characteristics of outgoing traffic change only *after* an internal host is infected.

Propagation: A firewall that has detected infection within its network using a local detection mechanism informs other firewalls of its infection. In our simplest scheme, a firewall can report only of its *own* infection: it cannot be implicated by other firewalls. This rules out *setup* attacks by which malicious firewalls make false accusations. When a detected firewall notifies other firewalls, other firewalls are said to be "alerted" to the attack. There are two forms of such notifications: *implicit* and *explicit* signaling.

In the implicit method, a detected firewall *marks* all suspicious outgoing packets. This marking serves two purposes. First, it informs the destination firewall that the packet is possibly malicious, and thus the packet should be dropped. Second, the destination firewall is also notified that the source network is infected. This can enable the destination firewall to install filters before its own network gets infected: this is the essence of cooperation. In explicit signaling, the firewall begins to send notifications of its infection to other participating firewalls. The detected firewall sends such notifications to other participating firewalls at some rate (perhaps varying the rate with the level of perceived infection). These notifications, both explicit and implicit, include some information about the worm attack: specifically, filters that can be used to identify malicious packets. Thus, every firewall is in one of the following four states (indicated in Figure 2): quiescent, alerted, infected and detected.

Filtering: An alerted firewall installs filters and drops malicious *incoming* traffic matching these filters. A detected firewall also installs such filters, and marks matching *outgoing* traffic. These filters are based on port numbers: any traffic on alerted ports is dropped. A port number filter is cheap to implement, and is resistant to polymorphic worms. In some cases, dropping all packets based on a port number might be too drastic even during an attack. For example, an organization might want its web server to be accessible to clients even during a worm attack. For such commonly used ports like the HTTP port, we assume that the port filter is also augmented with some content filters as well. This content filter can as simple as the URL in a HTTP request. The problem of deducing content filters from malicious packets is outside the scope of our work (Autograph [16] is a recent proposal on finding such content filters).

3.2 Local Detection

The goal of a local detection scheme is for a firewall to analyze its outgoing traffic to detect infection within its network. Our scheme is based on the observation that a legitimate connection has a much lower failure probability than a worm scan [18]. To detect an internal host infected by a TCP worm, every firewall maintains the number of failed outgoing connection attempts per *internal* host by observing whether it sees a SYN ACK packet in response to a SYN packet sent by the host. A UDP worm can be handled in a similar fashion by waiting for a response from the remote machine. Most UDP protocols require some sort of a response from the server (at least to acknowledge the request), and this response can be used to decide whether the connection succeeded or not.

Our local detection scheme maintains observations of connection failure rates per internal host. If the number of such failed connection attempts exceeds a given threshold T_f over a time window W_f, then the firewall marks that host as infected. This per-host infection state is timed out after a certain duration. If per-host state is not possible, the firewall simply maintains the aggregate number of failed connections over all its internal hosts, and maintains per-host state only for the heavy-hitter hosts. However, observe that once a single host has been identified as infected, the firewall cannot rule out the possibility that all hosts within its network are infected. For this reason, when more than a certain number N_f of its hosts are infected, then the firewall marks all of its hosts as infected. It is now said to be in the "detected" stage. We assume that N_f is chosen such that the probability of a false positive in local detection is low.

Our local detection scheme is simple and straightforward with low per-host cost. This detection scheme works only for scanning worms, and fails for other classes of worms such as DNS scanning worms, search engine querying worms. Other schemes can be used in our framework: *e.g.,* the Threshold Random Walk (TRW) scheme [19] to improve sensitivity or worm fingerprinting [20] to handle other classes of worms. The performance of our worm containment scheme depends only on the following metric of local detection: the number of scans an infected host can make before being identified. With other local detection schemes, it may be possible to improve the performance of our scheme.

3.3 Propagation

A detected firewall begins to signal other participating firewalls of its own infection. In the implicit signaling method, a detected firewall *marks* all outgoing packets on the attack port identified (or by using a content filter for commonly used ports). These markings include filters identifying the worm and can be implemented by using some bits in the IP header or by encapsulating the packet if required. In explicit signaling, the firewall directly sends notification to other firewalls at rate E. We assume that each detected firewall knows the identity of all participating firewalls. For now, we only consider explicit signaling schemes where each detected firewall sends signals to other firewalls at a constant rate E. This naive method can easily be improved by, say, a publish-subscribe based system. In such a system, the detected firewall need not send its own notifications to other firewalls: it can publish it to other firewalls who can send these notifications to interested firewalls. This improves the effective notification rate. The implicit method has the advantage that no new packets are generated: thus it has minimal overhead. However, if the infected host stops scanning after its firewall has detected infection, then implicit signaling will be of no use. Explicit signaling, on the other hand, has the other advantage that the propagation is independent of the scanning pattern of the worm.

We now describe three specific issues in implementing propagation. First, if a firewall has identified some hosts as infected but has not yet entered the detected stage (because the number of such hosts is less than N_f), then it can rate-throttle or simply drop outgoing connections from that host. It should not however propagate markings to other firewalls to avoid false positives at the global level. This helps deal with hosts that initiate port-scanning. Second, at the later stages of infection, the firewall can start dropping outgoing scans using the filter to detect such scans. This can be done without any impact on propagation if a signal has already been sent to the scan's destination firewall. Third, in both types of signaling, the destination firewall must be able to authenticate the source of the signal: otherwise a malicious end-host could simply generate fake notifications by using address spoofing. We use a simple challenge-response scheme for this purpose. Such authentication can be performed by simply contacting the allegedly infected firewall, and verifying that it indeed originated the signal.

3.4 Filtering

There are two types of filters installed in our scheme. The first is an outgoing traffic filter deployed at detected firewalls in order to mark matching packets (for sending implicit signals). The second is an incoming traffic filter established at all alerted firewalls. This filter drops marked packets: the source firewall is in the best position to decide whether its hosts are infected, and all such possibly malicious packets will be dropped. When a firewall receives an alert from another firewall, due to our trustworthy firewall assumption, it assumes that an Internet-wide worm attack is in progress, and drops all incoming traffic (marked/unmarked) matching the filter specified in the alert. Thus, once a firewall is alerted, *all* traffic matching the filter is dropped. Designing filters that accurately identify worm traffic with a low false positive ratio is outside the scope of this work: we use simple port-based filtering augmented with content filters for commonly used ports.

4 Modeling

We now analyze the effectiveness of the scheme proposed in the previous section in the complete deployment scenario. Our measure of effectiveness is the fraction of networks which escape infection at $t = \infty$: this is the *containment* metric (C).

N	Total number of Vulnerable Firewalls	$1,00,000$
n	Number of Deployed Firewalls	$1,00,000$
h	Number of vulnerable hosts per firewall	10
p	Successful probe rate	0.00025
s	Scanning rate of worm	4000 per sec
t_d	Detection time	0.2 secs
λ	Birth Rate	$phst_d$
s_n	Normalized Scan Rate	phs/N
σ	Defined for notational convenience	$\frac{\lambda-1}{t_d}$

Table 1: Default setting of parameters in our model

4.1 Parameters

The default parameters used to evaluate our solution are specified in Table 1. Apart from the network and worm model parameters, this table also includes other variables that will be defined later to aid in the analysis. Our network model and worm model are as specified in Section 2. The parameters were chosen based on the homogeneous cluster model of Slammer victims [21]. We set the scan rate to the average scan rate during Slammer. We set the total size of the vulnerable population to that of Blaster [5], one of the most wide-spread worms. We assumed 1 million vulnerable hosts divided equally among $1,00,000$ vulnerable networks. The probability of a successful probe p is the number of vulnerable hosts divided by the size of the IP address space.

Each deployed firewall sets a threshold of T_f failed connection attempts over a time window W_f. In our analysis, we assume that T_f is aggregated over all internal hosts: that is, the firewall maintains only an aggregate count of failed connections. Thus, if all the h vulnerable hosts behind a firewall are infected and scan at a rate s, the firewall will enter the detected stage, once it sees T_f failed connections within W_f seconds. The average number of failed connections seen per second by an infected firewall is $hs(1-p)$, and thus the firewall will detect infection within time $t_d = \frac{T_f}{hs(1-p)}$: we call t_d the *detection time*. We used a threshold of 800 failed connections per host: this gives $t_d = 0.2$. Note that it is advantageous for all the infected hosts in the network to probe at the same time: otherwise, the firewall would cut off the other hosts after identifying T_f connection failures.

We model the infection process as follows. At any point in time, there are four types of firewalls: infected firewalls have detected their infection, infected firewalls that have not yet detected their infection, uninfected firewalls that have been alerted by some detected firewall, and uninfected firewalls that have received no alerts. At time t, denote by $n_i(t)$ the number of infected firewalls, by $n_d(t)$ the number of detected firewalls, and by $n_a(t)$ the number of alerted firewalls.

4.2 Effectiveness of detection and filtering

First, we analyze a simplified variant of our scheme without any propagation (thus, there is no sharing of information between firewalls). In this variation, a firewall performs only local detection and filtering (once it has entered the detected stage, it installs a filter to drop outgoing scans). Thus, the only defense against a worm attack is that an infected host can only probe for time t_d (after which all its scans will be dropped by its firewall). The following lemma shows that such a simplified scheme is surprisingly effective (under some conditions). We first define $\lambda = hspt_d$ as the birth rate.

Lemma 1. *If $(\lambda < 1)$, then as $N \to \infty$, the containment metric $C \to 1$. Further, if I_0 and I_∞ denote the number of infected firewalls at time $t = 0$ and $t = \infty$ respectively, $E[I_\infty] = \frac{I_0}{1-\lambda}$.*

Proof. We prove this lemma using a differential equation model, which naturally extends to our analysis of propagation.

$$\frac{d}{dt}(n_i) = (hsp)(n_i - n_d) \quad (1)$$

$$\frac{d}{dt}(n_d) = \frac{(n_i - n_d)}{t_d} \quad (2)$$

The first equation follows since the rate of increase of infected firewalls is the same as the rate of successful scans (expressed as the number of infected undetected networks times the scanning rate per network). This equation overestimates the value of $n_i(t)$, since it assumes that no scans are sent to already infected firewalls. This overestimate suffices for this analysis. The second equation shows that the number of detected firewalls follows the number of infected firewalls with a phase lag of t_d (since it takes time t_d for detection). Defining $f(t) = n_i(t) - n_d(t)$, where $f(t)$ is the number of scanning firewalls, gives the following equation (by subtracting Equation (2) from Equation (1)):

$$\frac{d}{dt}(f) = (hsp - 1/t_d)i$$

If $\lambda < 1$, then $(hsp - 1/t_d) < 0$, and thus this corresponds to an exponential decay with a decay rate of $(1-\lambda)/t_d$. This implies that as $n \to \infty$, $C \to 1$. One can also identify this process as a birth-death process [22] in time steps of t_d to obtain $E[I_\infty] = \frac{I_0}{1-\mu(B)} = \frac{I_0}{1-\lambda}$ (proof omitted).

□

Corollary 1. *If $(\lambda < 1)$, then for any finite N, $C \geq 1 - \frac{I_0}{(1-\lambda)N}$ against any class of scanning worm*

Proof. Notice that in the previous proof, we used the assumption that $N > N_0$ to ensure that no two successful probes were sent to the same network. But, that is precisely what the ideal smart scanning worm would do in order to avoid wasteful probing. Thus any form of scanning worm can only cause a lower containment metric: thus $E[I_\infty] \leq \frac{I_0}{1-\lambda}$ and $C \geq 1 - \frac{I_0}{(1-\lambda)N}$. Note that a smart scanning worm might adopt strategies to increase the detection time t_d: however, given a specific t_d, the containment is expressed by the above equation. □

4.3 Effectiveness of Implicit Signaling

In this section, we analyze the containment metric when implicit signaling is used along with detection and filtering. There is no explicit signaling, and this models the case of a

worm that performs a topological infection followed by uniform random scanning. We extend the differential equation approach to account for implicit signaling as follows:

$$\frac{d}{dt}(n_i) = \frac{phs(n_i - n_d)(n - n_i - n_a)}{N} = s_n(n_i - n_d)(n - n_i - n_a) \quad (3)$$

$$\frac{d}{dt}(n_d) = \frac{n_i - n_d}{t_d} \quad (4)$$

$$\frac{d}{dt}(n_a) = \frac{hs(n_d)(n - n_i - n_a)}{N} = \frac{s_n n_d(n - n_i - n_a)}{p} \quad (5)$$

Here, s_n is the normalized scan rate phs/N defined for convenience. The first equation is the same as Equation (1) except that we exclude scans that are sent to alerted or infected firewalls. The second equation is the same as Equation (2). The third equation calculates the rate of spread of alerts as the number of alerts per detected firewall multiplied by the probability of the alert being sent to an unalerted (and uninfected) firewall. We could not solve these equation exactly, so we present two additional results: a closed-form upper bound on the containment metric and numerical integration of these equations. Our upper-bound is given in the following lemma:

Lemma 2. *For $\lambda > 1$, the containment metric C by using implicit signaling is at least $1 - \frac{(log(N)+c)t_d\sigma^2)}{hs}(\frac{1}{t_d\sigma}+1)$ with probability of at least $1 - e^{-e^{-c}}$, where $\sigma = (\lambda - 1)/t_d$.*

Proof. We derive a lower bound on the containment metric by dividing the worm infection into two phases: the first phase extends until the time all firewalls have been alerted, at which point the second phase begins. In the second phase, the worm cannot spread any further since all firewalls have been alerted. We approximate the first phase as follows: we assume that no firewall has been alerted, and thus the worm propagates just as in the case of detection and filtering alone. Clearly, this suffices to derive an upper bound on the number of infected machines at the end of the first phase, and consequently a lower bound on the containment metric. We first analyze the progress of the worm in the first phase (there are no alerted firewalls, only detection and filtering). This is the same as Equations (1,2):

$$\frac{d}{dt}(n_i) = \frac{(hsp)(n_i - n_d)(n - n_i)}{N} \approx (hsp)(n_i - n_d) \quad (6)$$

$$\frac{d}{dt}(n_d) = \frac{(n_i - n_d)}{t_d} \quad (7)$$

The first equation approximates $(n - n_i)/N$ to 1 since an upper bound of n_i suffices for our purposes (we assume that the number of infected firewalls remains low throughout the first phase or equivalently that the worm does not probe infected networks). This implies that:

$$\frac{d}{dt}(n_i - n_d) = (phs - \frac{1}{t_d})(n_i - n_d) = \sigma(n_i - n_d)$$

$$n_i(t) - n_d(t) = I_0 e^{\sigma t} \quad (8)$$

The first equation uses $\sigma = (\lambda - 1)/t_d$, where λ is the birth rate. We assume $\lambda > 1$ since otherwise there is no necessity for implicit signaling. The second equation follows by integrating the first, and imposing the constraints that $n_d(0) = 0$ and $n_i(0) = I_0$, the number of infected firewalls at $t = 0$. This can be substituted in Equations (6), (7) to derive n_i, n_d as a function of time:

$$\frac{d}{dt}(n_d) = \frac{I_0 e^{\sigma t}}{t_d} \Rightarrow n_d = \frac{I_0(e^{\sigma t} - 1)}{t_d \sigma} \quad (9)$$

$$n_i = I_0 e^{\sigma t}(\frac{1}{t_d\sigma} + 1) - \frac{I_0}{t_d\sigma} \quad (10)$$

The first equation is obtained by integration, and the second by substitution in Equation (7). These two equations track the growth of the number of infected and detected firewalls in the first phase. Based on this, we calculate the *total* number of implicit signals $i(t)$ sent by time t as:

$$i(t) = \int_0^t (hs) n_d \, dt = \int_0^t \frac{I_0 hs}{t_d\sigma}(e^{\sigma t} - 1) = \frac{I_0 hs}{t_d\sigma}(\frac{e^{\sigma t}}{\sigma} - t) \quad (11)$$

The allocation of these $i(t)$ implicit signals sent by time t to the various firewalls is analogous to the allocation of coupons in the classical coupon collector's problem [22]. Note that the first phase ends when all firewalls have received at least one alert each, analogous to the condition that all coupons must be collected. Notice here that since the worm scans external networks uniformly at random, implicit signals follow the uniform distribution as well. We use the standard result that if m is the number of coupon collectors and the number of coupons exceeds $m\,log(m) + c\,m$, then the probability that all m collectors have one coupon each is at least $1 - e^{-e^{-c}}$ [22]. In our case, since each firewall must receive an alert to get alerted, we use $m = N$. Thus, the time t_1 at which the first phase ends with probability $\geq 1 - e^{-e^{-c}}$ can be calculated using $i(t_1) = N\ log(N) + c\,N$ (we approximate $\left(\frac{e^{\sigma t}}{\sigma} - t\right)$ to $\frac{e^{\sigma t}}{\sigma}$). This time t_1 can be substituted in Equation (10) to obtain $n_i(t_1)$, the number of infected firewalls at the end of the first phase as:

$$n_i(t_1) = \frac{N t_d \sigma^2 (log(N) + c))}{hs}(\frac{1}{t_d\sigma} + 1) \quad (12)$$

and thus $C = 1 - \frac{n_i(t_1)}{N}$ with probability $\geq 1 - e^{-e^{-c}}$. □

Implications: We substituted the different parameters from Table 1 to obtain the upper bound. σ is calculated to be 5 and we choose c so as to reduce the error probability to less than 10^{-3}. We get $C \geq 97.3\%$, which is very effective. Thus, implicit signaling works well under the trustworthy firewall assumption. Note in this case, that $\lambda = 2$, which means that detection and filtering would not work against this worm. This lemma also tells us how the containment metric behaves with respect to the various parameters. Note that t_d varies with s as $t_d s = k_1$, a constant (since $t_d = \frac{T}{hs(1-p)}$). Thus, C can

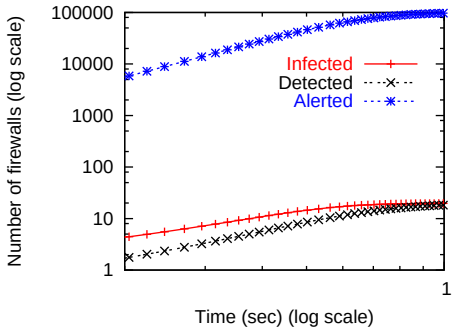

Figure 3: Dynamics with Time

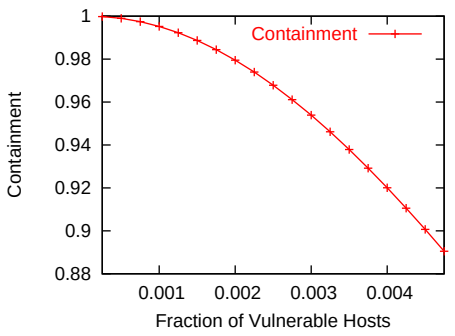

Figure 4: Containment versus varying vulnerable population sizes

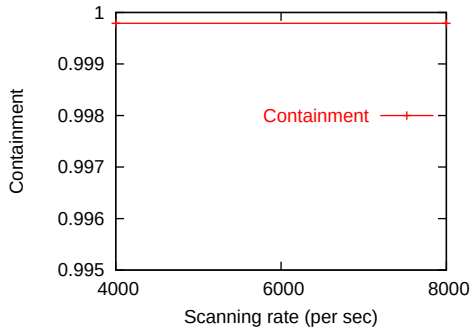

Figure 5: Containment versus scan rate

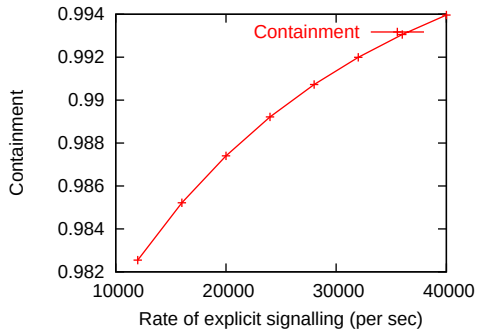

Figure 6: Containment versus explicit signaling rate

be simplified to $1 - log(N)p^2hk_1(\frac{1}{k_1hp} + 1)$ at high scanning rates by approximating $\sigma = phs$. Note that implicit signaling along with our local detection scheme is unaffected by the scanning rate of the worm, since the rate at which signals are sent increases linearly with the scanning rate of the worm. Also observe the containment metric decreases as $log(N)$ and quadratically with respect to p. This highlights one of the main disadvantages of implicit signaling: the rate of signaling is limited by the scan rate of the worm itself and these scans can only be sent in a random fashion.

4.3.1 Numerical Solutions

In order to understand the properties of our propagation scheme and local detection separately, we first study our propagation scheme at various values of t_d (the time for a firewall to enter "detected" stage once all its hosts are infected). The local detection scheme determines the value of t_d as $\frac{T}{hs(1-p)}$. The detection time t_d is lower at higher scan rates, so our local detection schemes detect faster worms faster.

Temporal Dynamics: The dynamics of the number of infected, detected, and alerted firewalls over time is plotted in Figure 3. The the number of detected firewalls always lags behind the number of infected firewalls as expected. Very early in the propagation, the number of alerted firewalls surpasses the number of infected firewalls, and from then on, grows like the classic sigmoid curve. This is because we are using implicit signaling, in which case the spread of alerts is similar to that of the worm itself.

Effect of number of vulnerable hosts: The containment metric of implicit signaling is plotted against the size of the vulnerable population in Figure 4 (b). We increased the number of vulnerable networks as we increased the size of the vulnerable population (such that h was constant). Again, as suggested by the analysis, there is roughly a quadratic drop in C with p.

Effect of Scanning Rate: The performance of implicit signaling against the scanning rate of the worm is shown in Figure 5. Along with the scanning rate, t_d was also varied such that their product was constant (as mentioned, before this is a property of our local detection scheme). As suggested by the analysis, the performance of implicit signaling is independent of the scanning rate of the worm: there is no variation despite a two-fold increase in the worm scan rate.

4.4 Effectiveness of Explicit Signaling

In the naivest form of explicit signaling, each detected firewalls send signals at a constant rate of E per second to randomly chosen firewalls. In this case, the containment metric can be obtained in a similar fashion to the derivation in the previous section: the result can be obtained by simply substituting $hs = hs + E$ in the lemma to obtain $C \geq 1 - \frac{(log(D)+c)t_d\sigma^2)}{hs+E}(\frac{1}{t_d\sigma} + 1)$. This analysis can be further made tighter since explicit signals will not be sent to the same firewall twice. Even in this naive method, setting $E = s$ (same as the scanning rate of the worm), improves the containment metric to above 98.5%. Thus, this formula can be used to compute the rate of explicit signaling required to achieve a given containment metric. This same extension can

be used in the analysis in Equations (3,4,5). This analysis also indicates that in order to maintain the same containment metric with increasing number of firewalls, the rate of explicit signaling need only grow logarithmically $O(\log(N))$.

Effect of rate of explicit signaling: The containment metric of explicit signaling is plotted against the rate E in Figure 6. It shows the variation as E is varied (in multiples of the worm scan rate). The containment metric improves with the ratio E/hs, and one can achieve a desired containment metric by increasing the rate of explicit signaling.

4.5 Non-uniform host distribution

We now show that our analysis is also useful in the case when the distribution of vulnerable hosts is non-uniform. If most of the vulnerable hosts live only in a few networks, then one can approximate it by defining h based on the number of vulnerable hosts in those networks, and defining p based on the global density in the 32-bit IP address space. It is also possible that the distribution of vulnerable hosts is uneven even among the vulnerable networks. In the case of Slammer, this was indeed the case [21]. Our analysis also sheds light on the effectiveness of our scheme under such a distribution. In this case, note that the performance of implicit signaling does not depend on the distribution. This can seen from Equations (3, 5): the differential $\frac{d}{dn_i}(n_a)$ is independent of h, the number of vulnerable hosts in each network. This differential, in fact, does not depend on the number of vulnerable hosts in each individual infected network. Thus, one can derive $n_a = g(n_i)$ where g is a function independent of the distribution of vulnerable hosts. The containment metric C can also be obtained by solving $n_a = g(n - n_a)$ where $n_a = Cn$, since at $t = \infty$, all networks are either infected or alerted. Thus, although the *time for containment* would depend on the distribution, the *containment metric* itself would not. In the case of explicit signaling, the ratio E/hs determines the containment: in this case, a firewall with a greater number of infected hosts can send explicit signals at a higher rate. Thus, our schemes and analysis can also be applied to deal with the non-uniform distribution of vulnerable hosts among vulnerable networks. However, note that there are other effects in the Internet that are not captured by our model such as heterogeneous access link capacities noted in the case of Slammer [21].

4.6 Summary

In this section, we have shown that for different ranges of $\lambda = phst_d$, different measures are effective. Detection and filtering is the simplest scheme: there is no need for coordination between multiple firewalls and scans are dropped at the source firewall after detection. Even this simple scheme works in the regime $\lambda < 1$. However, there is the possibility for faster worms with $\lambda > 1$. For such worms, implicit propagation gives good containment under the trustworthy firewall assumption in the complete deployment scenario. This containment can be further improved by adjusting the rate of explicit propagation to achieve the desired containment level. This analysis also serves to highlight the tradeoffs between implicit and explicit signaling.

Our analysis also reveals the sensitivity of containment to the various parameters in the model, which can be helpful in identifying ways to improve containment. For example, our analysis can be used to determine the granularity at which firewalls should be deployed. It also helps us to determine how sensitive a local detection scheme should be, so that a desired level of containment can be attained. We note that our simple local detection scheme seems to have low enough detection time t_d so that good containment is possible. Our local detection scheme uses failed connection attempts to find infected hosts, and can detect faster scanning worms in a shorter time. This variation can be described as $t_d s(1-p) = \text{constant}$, where the constant depends on the desired probability of false negatives. We have used $t_d = 0.2$ for a worm with a scan rate of 4000 per second, using the threshold for failed connections as 800.

Also note that our analysis can be extended for classes of worms other than scanning worms. The use of an explicit detection time t_d allows the analysis of the propagation scheme to be decoupled from local detection. Our local detection mechanism works only for fast scanning worms, and that fact is used to derive t_d. The above analysis also works for other kinds of worms and suitable local detection. The only constraint is that local detection must be able to detect an infected network within t_d seconds. For example, a worm that uses a pre-generated hit list can be detected by say, a scheme counting the number of unique destination IP addresses in the outgoing traffic. In this case, one would also set $p = 1$ since every scan is successful. Our analysis can then be used to determine how low the detection time t_d must be for a given level of containment: this can be used to evaluate local detection schemes.

5 Partial Deployment

In this section, we remove the requirement of complete deployment, and handle the scenario when not all firewalls in the Internet participate in our cooperative. We first propose a simple technique that works in the partial deployment scenario, and then describe a technique called *rerouting* to improve the containment metric.

5.1 Baseline Solution

As a first cut, we observe that our existing scheme for propagation works even without assuming local detection at every firewall. In this scenario, only the deployed firewalls perform detection and propagation: undeployed firewalls do not engaged in local detection, propagation, and installing filters. All the undeployed firewalls will be infected eventually (since an infected undeployed network can always probe other undeployed networks). The only benefit for such networks is that the worm propagation time is increased since deployed firewalls drop outgoing scans (after detection).

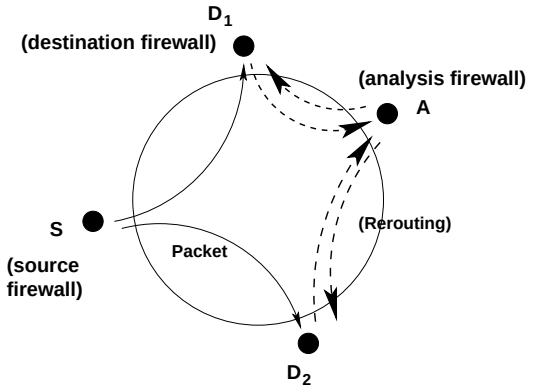

Figure 7: Rerouting

5.2 Rerouting

The basic idea to improve the effectiveness under partial deployment is to emulate the idealized model using *rerouting*. Rerouting is similar to the concept of wormholes in [12] with the main difference that we employ it in a decentralized context. We first describe rerouting, and then elaborate on how we use rerouting to improve performance under incremental deployability. We mention the main limitation of rerouting when used in the incremental deployment scenario: it works only for a TCP worm or a bi-directional UDP worm. We assume that the firewalls in our cooperative can verify whether this condition holds true and invoke rerouting if so.

So far, we have relied on the source firewall (the firewall of the source's network) to monitor its outgoing traffic, detect infection and then initiate signaling. This is not possible if the source firewall does not implement our scheme. For this reason, we use a rerouting mechanism that allows other firewalls to monitor their *incoming* traffic and attempt to detect infection of the source firewall. This rerouting mechanism is based on a mapping (say, based on a hash function) from a *source* firewall to an *analysis* firewall. All firewalls can compute this mapping. All deployed firewalls redirect incoming traffic coming from firewall X to the analysis firewall for X, denoted by $A(X)$. Here, X is an undeployed firewall, and $A(X)$ is a deployed firewall. We refer to this mechanism as *rerouting*. This is similar to receiver controlled redirection proposed in overlay networks literature (for example, in *i3* [23]).

The main idea of rerouting (illustrated in Figure 7) is that the analysis firewall $A(X)$ for X can perform local detection and propagation on behalf of X. If all firewalls redirect their incoming traffic from X, then all traffic sent by X is seen at $A(X)$. This means that the firewall $A(X)$ can exactly mirror the state that would be maintained at X. Since only deployed firewalls would forward traffic from X to $A(X)$, the traffic seen by $A(X)$ does not include traffic sent by X to other *undeployed* firewalls. Thus, the traffic seen by $A(X)$ serves as a crude sampling of the traffic sent by X. Thus, $A(X)$ can carry out detection and propagation on behalf of X (of course, it is up to X itself to filter its incoming traffic). Our local detection scheme requires per-host state regarding connection failure rates: this state is available at $A(X)$ which can thus detect the infection of network X using the same test. $A(X)$ can also propagate signals on behalf of X. Implicit signals can be propagated since $A(X)$ can mark a redirected packet before returning it to the destination firewall. Similarly, it can also propagate explicit signals on behalf of X. Other firewalls can verify such a signal by ensuring the source of such a signal is indeed the analysis firewall for X.

Thus, each *deployed* firewall acts as an analysis firewall for a set of *undeployed* firewalls, and maintains state and performs propagation on their behalf. This improves the containment metric because scans from undeployed firewalls will also be detected and used to initiate propagation. Note however that this scheme only monitors the scans sent by an undeployed firewall to some deployed firewall: probes from an undeployed network to another are not routed through an analysis firewall. Thus, this is only an approximation of the idealized scenario when all firewalls are deployed. Another respect in which rerouting differs from the idealized scenario is as follows: if it is possible for the worm to spoof source addresses, then rerouting does not work since an infected host in an undeployed network can spoof the address of a host in another undeployed network. TCP worms or UDP exploits that require the host to send a response cannot spoof the source address: only single packet UDP worms can do do. Of course, if the undeployed networks implement egress filtering, this can be avoided.

This scheme requires the overhead of redirecting every packet to an analysis point: in practice, this may be avoided by redirecting only connection setup packets. The filtering itself can be performed at the destination firewall: the analysis firewall only marks the packet. Two datasets used in TRW [19] in fact offer evidence that the number of connection setup packets may be low enough to implementing such rerouting with manageable overhead. In the first dataset, the number of inbound connections to 217 hosts (living in an address space of 512 hosts) is about $160,000$ over one day, which means an average of about 0.008 new connections per second per host. In the second dataset on a much busier network, the number of inbound connections to about 6000 hosts (in an address space of about $130,000$) is 15.6 million, which amounts to an average of 0.003 new connections per second per host. Assuming a 40-byte TCP SYN packet, rerouting amounts to less than 1 KB/s average bandwidth even in the busier network. This suggests that rerouting connection setup packets may be feasible in practice: this assumption needs to be verified further by more traces. Note however that connection setup will be delayed by the round-trip delay. Other optimizations may also be possible in practice, such as, sampling or sending summaries of connections instead of sending notification about every single connection.

5.3 Numerical Results under partial deployment

Unlike the complete deployment case, note that detection and filtering will not work in any regime under partial deployment (thus, Lemma 1 does not hold). We have derived a differential equation model for the partial deployment scenario in Appendix A. We now illustrate numerical results based

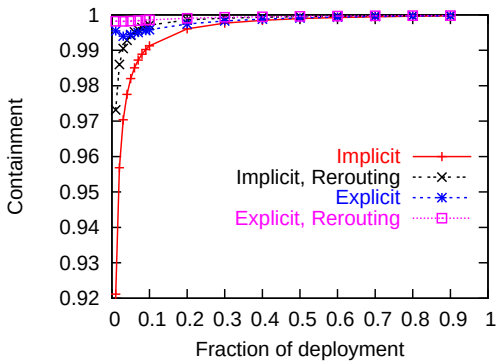

Figure 8: Containment versus Deployment

on this model. As before, all parameters are chosen from Table 1. For a fixed number of total firewalls, the graph in Figure 8 shows the variation of the containment metric with fraction of deployed firewalls. On this graph, we show four plots: one with implicit signaling, one with implicit signaling along with rerouting, one with explicit signaling, and one with explicit signaling combined with rerouting. In the last two cases, implicit signaling was also used. The explicit signals are sent at rate E which is set to 10% of hs, the scanning rate of an infected network. Even without rerouting, implicit signaling perform reasonably well. This is because the containment metric for implicit signaling depends only on $log(n)$, and thus there is not much drop in the performance. Thus, even against a single packet UDP worm, our schemes provide good containment (over 92% under 1% deployment). Rerouting helps to improve the performance when combined with implicit signaling: in the 1% deployment scenario, it improves the containment metric to over 97%. Explicit signaling performs very well even under 1% deployment offering over 99% containment. The main reason for this is that explicit signals are sent to only participating firewalls, unlike implicit signals which will be wastefully sent to all firewalls.

6 Security

In this section, we perform a security analysis of our scheme against various kinds of attacks and discuss defensive measures. There are three types of security attacks in our system depending on the perpetrator: malicious end-hosts, malicious firewalls, and smart worms. The second category also includes false alarms triggered by incorrect information propagated by misconfigured firewalls (that do not have malicious intent).

The first category of attacks is easy to defend against in our system. Any signaling initiated by an end-host will be rejected (we assume that all firewalls know each other's address, and as mentioned before, a challenge-response mechanism precludes address spoofing attacks). An end-host can attempt to trigger fast alarms by sending scans. If the number of such end-hosts within a firewall is less than N_f (the threshold for the number of infected hosts required to enter the detected stage), then the firewall will only block traffic generated by such malicious end-hosts. Otherwise, the firewall will itself trigger a false alarm, and we treat this firewall as a malicious firewall (and such attacks will be considered in the second category). Thus, an end-host has limited ability to influence the detection capability or the propagation. Attacks by malicious firewalls and smarter worms are harder to deal with: these will be the subject of the next two sections.

6.1 Malicious Firewalls

Some of the firewalls in the cooperative could be malicious and attempt to trigger false alarms or suppress signals. We only attempt to deal with the case when there are a few such firewalls in the cooperative: cases where firewalls themselves are subverted by a worm are beyond the scope of this work.

First, note that in our scheme, one firewall cannot implicate another: a firewall can only initiate signals (implicit/explicit) about itself. Of course, this requires that the source of every signal be verified by the recipient firewall using a challenge-response mechanism. In the case of implicit signals, the recipient can request verification from the allegedly infected firewall, and drop the signal if the latter does not confirm the notification. The same can be done for explicit signals as well. In the partial deployment scenario, only the analysis firewall $A(X)$ of X is authorized to send signals about X: signals sent by other firewalls implicating X will be rejected. Note however in this case, that a malicious firewall M can route a SYN packet to $A(X)$, spoofing the source address of the packet to X. $A(X)$ would then notice that there was no response to this packet, and would incorrectly conclude that X was originating the packet. If X were deployed, $A(X)$ could verify by asking X whether it was indeed was the source, before initiating markings: however, if X is one of the undeployed firewalls, this protocol does not work. One straightforward solution to this problem is that $A(X)$ wait for reports of connection failures from several firewalls before considering X to be infected. Note that this issue arises only in verifying signals from analysis firewalls that monitor undeployed networks.

This leaves us to consider the case when a malicious firewall propagates incorrect infection about itself. There are two kinds of such attacks: the firewall can claim to be infected and thus aim to trigger an Internet-wide false alarm, or the firewall can suppress notifications when itself infected. Regarding the first class of attacks, note that a malicious firewall can always pretend to be infected by originating scans on its own. It is not possible to distinguish between such a malicious firewall and a truthful firewall with infected hosts. Thus, in a large cooperative, there will be a few firewalls that can behave maliciously to trigger such scans, and our solution to this problem is to require that each firewall should receive D alerts from different firewalls before entering the alerted stage. Such a scheme can resist up to D malicious firewalls behaving in such a fashion. Our aim is to resist such attacks from a few hundred firewalls, beyond which, auditing by other firewalls may be used to discourage such behavior. We discuss the implications of this modification in Section 6.1.2.

6.1.1 Redundant Rerouting

We now discuss how the rerouting mechanism can be extended in order to deal with the second class of attacks. The basic idea is that there are m analysis firewalls associated with every source firewall. Note that this mapping from source firewall to a set of analysis firewalls is based on a hash function. The simplest form of such redundant rerouting is as follows: each deployed firewall sends a copy of an incoming packet from source firewall X to each of these m analysis points, and delays the packet (by storing in a local buffer). Each of these analysis firewalls notifies it (implicit/explicitly) of its decision on whether the source network is infected. The firewall now takes the majority consensus of these decisions, and decides whether to admit such a packet or not. Of course, all these packets can be sent in parallel to the m analysis firewalls, so only a single round trip will be incurred. Moreover, as discussed before, only connection setup packets need be forwarded, and sampling techniques can be used.

6.1.2 Implications

We now examine the containment metric under the condition that D distinct alerts have to be received by a single firewall to get alerted. An exact analysis, similar to that in Section 4.3, can be made by using $(D+1)$ differential equations to track the number of firewalls that have received $0, 1, \cdots, (D-1), D$ alerts. This analysis amounts to a Markov chain model, where the transition probabilities vary with time: this makes the differential equation method very cumbersome. The analysis used to obtain the lower bound on containment metric can however be applied to derive:

Lemma 3. *For $\lambda > 1$, the containment metric C by using implicit signaling is at least $1 - \frac{(Dlog(ND)+cD)t_d\sigma^2)}{hs}(\frac{1}{t_d\sigma}+1)$ with probability of at least $1-e^{-e^{-c}}$, where $\sigma = (\lambda-1)/t_d$.*

The only change required is that the number of alerts required before the first phase finishes, has to be at least $(NDlog(ND) + cND)$ since each of the N firewalls has to receive D distinct alerts. For the default values we have been using (from Table 1), we now set $D = 10$ malicious firewalls. We get $C \geq 77\%$, which is still effective. Observe that C decreases with $Dlog(D)$, and cannot deal with more than a few hundred malicious firewalls. This analysis can also extended to include explicit signaling by replacing hs with $(hs + E)$. The rate of explicit signaling can be adjusted to deal with malicious participants. This required signaling rate scales at the rate of the number of malicious participants D that we wish to resist (for $D \ll N$, $log(ND)$ grows slowly with D, and can be considered to be a constant). Thus, we can resist, say 100 malicious firewalls, by setting $E = 10hs$ for a containment for 77% (since we can resist 10 malicious firewalls for the same level of containment with no explicit signaling).

The parameter D can be increased *without* affecting C as follows: it is possible for firewalls to form sub-cooperatives of size k among themselves. The firewalls in such a sub-cooperative agree to forward all signals to one another, and enter the alerted state upon collectively observing kD distinct alerts. Thus, this scheme resists up to kD malicious firewalls without affecting C (assuming $kD \ll \sqrt{CN}$ to avoid duplicate markings by using the birthday paradox). Note again that alerts can always be verified in our system, and thus, even if there are malicious participants in the sub-coalition, they can only slow down detection by dropping alerts: they cannot trigger any false alarms. Thus, it is possible to use other techniques to increase the effective D, without affecting C. This technique can handle about a thousand malicious nodes with modest group sizes k (of say, a few hundred).

Note that redundant rerouting itself can be easily made resilient to malicious firewalls with low values of m. If there are D malicious nodes, the probability of more than $m/2$ firewalls being malicious can be derived using Chernoff's bound (for low values of D/N): $e^{\frac{-mN^2}{8D(N-D)}}$. This can be made very low by modest values of m (≤ 5).

6.2 Evasive Worms

We now consider worms that specifically attempt to attack our scheme (*e.g.*, do some form of "smart" scanning). We first discuss worms that attempt to evade local detection, and the move on to worms that attempt to subvert propagation. Note that filtering techniques may be susceptible to polymorphic worms [24] or worms that use a different exploit in successive phases of its attack: such attacks are beyond the scope of this work.

6.2.1 Evading Local Detection

The performance of our scheme clearly depends on the sensitivity of the local detection mechanism. The first observation that we make is that a typical *fast* scanning worm cannot evade local detection. The simplest technique to thwart detection is to scan at a rate similar to legitimate traffic: assuming that a legitimate host has about 1 failed connection attempt every 10 seconds, a worm that scans at this rate requires about 4.6 days to infect a vulnerable population as large as that of Slammer, and about a day to infect a population as large as that of Blaster. A worm that scans faster than this rate will trigger off the local detection scheme. Note that this can be improved by using better local detection schemes suggested in TRW [19], Worm fingerprinting [20]: as research advances in such schemes, they can be plugged into our model. This means that if a set of hosts behind a firewall are infected by a fast scanning worm, there will not be any false negatives in the local detection. Of course, slow worms can evade our local detection: such worms are beyond our consideration. Second, observe that it is not possible for a malicious host in an *external* network to make a firewall conclude that its network is infected. A firewall observes only *outgoing* traffic to decide whether it is infected. These two observations mean that our local detection scheme can be designed to have low false negative and low false positive ratios. False positives can only occur with a malicious internal host which behaves like an infected host: this is controlled by the parameter N_f.

Second, there could be worms that choose the destination of their scans by some mechanism other than scanning. They could rely on pre-generated hit lists or DNS scanning etc: we assume that suitable local detection schemes can be found

to deal with such worms and can be plugged in. Since our work decouples the efficacy of the local detection scheme from propagation itself, our results may be of use to handle other types of worms as well.

6.2.2 Evading Propagation

Next, we move on to attempts by worms to influence propagation. First, consider worms that aim to alter their scanning pattern over time in order to thwart propagation (but still choose their destinations randomly). Such worms can influence the propagation of our signals. This points to one of the chief advantages of explicit signaling: unlike implicit signaling which piggybacks signals on scans, explicit signaling does not. Thus, in the limit, a worm could simply stop scanning after its firewall has entered the detected stage, since beyond this point, all its scans will be marked and cannot infect any new hosts. In this case, implicit signaling is completely ineffective. Note however that setting the rate of explicit signaling E to the scanning rate of the worm s has the same effect as implicit signaling over a constant scan rate worm. This implies that explicit signaling can always be relied upon to handle such smart worms (our earlier analysis applies in this case).

Second, consider worms that choose their destination by using some smart algorithm (such as hit-list based) or by using coordination among themselves. For example, hosts in one firewall could agree to probe only 10 other networks. In this case, implicit signaling is adversely affected. However, explicit signaling still works: once such a worm is detected, explicit signals will continue to be sent *randomly*, which ensures that the signaling rate will not be slower than of implicit signaling containing a random worm. Note however that the infection pattern of the worm will not follow the analysis, and is dependent on its scanning algorithm.

Third, note that in the partial deployment scenario, there is a potential for worms that operate in two phases as follows: they coordinate amongst each other to first infect all the undeployed firewalls, and then all these infected firewalls begin to scan the deployed firewalls. Thus, until the beginning of the second phase, the deployed firewalls would not see any scans and would have no way to detect such a worm. Even in this case, our analysis can be used to show that with 10% deployment, about 50% containment can be achieved against such a worm (the containment was obtained by solving the differential equation model for partial deployment in Appendix A, using the initial condition that all undeployed firewalls are infected at time $t = 0$).

7 Related Work

There are several types of worms known in literature [8], and this work only deals with fast scanning worms. Other types of worms include those based on a pre-generated hit list or those that spread by consulting global directories. As we have mentioned before, it might be possible to extend our techniques to handle such worms by modifying the local detection scheme suitably: our results on propagation still apply. In general, dealing with scanning worms is considered a hard problem: a paper on Internet Quarantine [9] suggests that most known defense mechanisms such as patching, filtering might not work against very fast worms. Dealing with known worms is easy since signature based schemes (implemented in Snort [25], Bro [26]) can effectively contain non-polymorphic known worms. This work deals with zero-day attacks in which the vulnerability exploited by the worm is not known in advance.

The problem of defending against Internet worms has attracted considerable research effort, which we classify into four main categories. The first category includes works that attempt to use detection mechanisms at *end-hosts* in order to detect malicious packets. Stackguard [27] deals with software-based mechanisms for identifying stack smashing attacks, while there is also work on identifying malicious packets by tracing the pattern of system calls made by a process [28]. Vigilante [29] discusses leveraging such end-host based mechanisms in order to detect and spread information about worm attacks. Rate-based throttling [30] of initiation of connections to new IP addresses has been suggested on a per-host basis in order to increase the worm propagation time. In general, such host-based mechanisms can detect a wider class of worms since they have access to a more detailed level of information. However, in comparison with firewall based solutions (such as ours), they involve a heavy initial deployment cost: every end-host in a network has to be appropriately instrumented for protecting the network. Indeed, that is one of the chief reasons why firewalls have been so successful: they can transparently protect an entire network.

The second category consists of detection schemes that operate at a *single observation point* (not at the end-host) in order to detect a worm attack in progress. These can serve as local detection schemes in our framework. Methods to detect port scanning can be adapted to detect address scanning worms as well: *e.g.,* Spice [31] uses statistical anomaly detecting techniques to cluster suspect packets into portscans. Threshold random walks [19] is a more recent proposal where a hypothesis testing model is used to identify portscans, and this scheme has been extended for fast detection of scanning worms in [18]. Worm fingerprinting [20] aims to identify signatures based on flow characteristics observed using lightweight mechanisms operating at core routers: such characteristics include identifying identical traffic, diverse IP addresses, and failed connections. This scheme has been extended to design an early bird worm detection system [32]. Honeypots have also been proposed for early detection of worms (Honeycomb [33]) and there also have been proposals to build such honeyfarms by implementing several virtual honeypots on a single machine [34]. Network telescopes [35] proposes methods for tracking an Internet-wide worm attack based on observations of packets sent to a large unallocated address space. In general, these mechanisms offer the potential of very high sensitivity (with commensurate monitoring load as well), and are suitable for detecting worms at a single observation point. We view such work as orthogonal to our work. It is possible to envision a cooperative scheme where some of the participants can be honeyfarms dedicated for the purpose of detection.

The third category of work relates to proposed architectures for worm detection and containment involving *multiple vantage points*. There are two classes of such architectures proposed in literature: the first consists of architecture that perform distributed data collection followed by centralized analysis and second deals with decentralized architectures where all participants are implicitly trusted. Weaver et al [12] propose the use of firewalls that redirect traffic via virtual links (called *wormholes*) to a few designated honeyfarms. These honeyfarms consist of several honeypot machines, that can detect malicious payload. The concept of wormholes has also been in commercial use (*e.g.*, Counterpane [36]). Wu et al [37] propose an architecture where monitoring points within the network convey statistics to specialized analysis points. These statistic are used to identify infected hosts and to black list their address. Zou et al [38] propose Kalman filtering based methods to detect a worm attack based on observations from several monitoring points. This work also deals with statistical techniques for estimating the size of the vulnerable population and the infected population. NetBait [39] is a distributed query processing that provides users the ability to pose queries to identify infected hosts. GrIDS [40] aims to detect worm attacks by analyzing correlations in packet data collected from several monitoring points. Worm infection usually has a distinctive causal pattern of traffic which GrIDS aims to detect. Thus, these solutions [12, 36, 37, 40] are all based on the paradigm of multiple monitoring points redirecting traffic to a few dedicated analysis points, and thus fall in the first class of distributed architectures. While such a paradigm may be useful in certain contexts, we believe that the decentralized option also holds promise since the analysis and detection load is shared equally by all firewalls. This makes it harder to launch directed attacks against the worm detection architecture itself. The second class of distributed architectures consisting of trusted participants is very suitable for protecting an enterprise network. Containment within an organization network was proposed by the use of hard perimeters to prevent infection from spreading between LANs [10]. This work discussed issues in implementing worm protecting in hardware and is concerned with protecting a single organizational network by improving the granularity of monitoring points within the organization. Weaver et al [13] propose methods for very fast containment within an organization that rely on communication between "cells" that trust each other. This class of distributed architectures has also seen commercial applications (*e.g.*, CounterMalice [41]). Staniford [42] discusses containment within an organizational network by partitioning it into multiple compartments. They also identify a sufficient condition for worm containment in their framework, which is the same as our condition (Lemma 1 in Section 4) for containment under detection and filtering. Domino [11] is a more decentralized architecture where the analysis locations are organized in a overlay. Domino uses a overlay only to distribute the analysis operation: other monitoring points feed traffic to this overlay. It mainly focuses on detection, and does not deal with malicious participants. These solutions [10, 13, 41, 11] assume implicitly that the participants trust each other: our work aims to extend such cooperation schemes to the Internet where this assumption may not hold true.

The last category of work relies on the filter model for characterizing worms and for containing them. A filter is a method of deciding whether a given packet is possibly malicious worm payload. Autograph [16] attempts to infer a content filter based on observations from several monitoring points. It is based on the intuition that during a worm attack, several packets contain similar looking content (the exploit code itself). The efficacy of deployment of such filters in a Internet-like graph is studied by Park et al [43]. Their finding is that due to the power-law nature of the Internet, if filters are deployed in the high degree nodes, then worm containment can be achieved. The effectiveness of such filters when deployed at other locations within the network is considered in dynamic quarantine [44]: this work demonstrates that if deployed at backbone routers, rate throttling can be very effective. However, the impact of such throttling on legitimate traffic, especially diverse traffic such as peer-to-peer applications has not been studied in practice. DShield [45] deals with the placement of honeypot like code filters in the network: specialized points in the network which run the code found in packets to detect malicious packets. A architecture for automatically detecting malicious packet and generating vaccines for immunizing vulnerable end-hosts has been proposed by Sidiroglou et al [46, 47]. Other proposals that rely on router support for detecting and containing a worm include a ICMP redirection based scheme [48] and DEWP [49]. Berk et al [48] suggest that ICMP "destination unreachable" messages be redirected from some routers to a centralized analysis point, which can then detect a worm attack. In DEWP [49], routers analyze the bi-directional traffic flowing through them to identify port numbers that appear frequently in both directions. In general, filter placement within the Internet core seems to be necessary for some of these schemes. The feasibility of such placement is not very clear, since routers may be hard to modify and access to in-network processing boxes is usually limited. Our work deals with the propagation of such filters amongst firewalls at the edges of the Internet, and advances in filter representation can be used in our framework.

Finally, there have been two recent proposals for cooperative detection and containment in an untrusted environment. This problem has also been recognized by Sandin [50], who also posed the problem of designing a peer-to-peer based system for this purpose. Nojiri et al [15] propose the use of a "friend" network: each firewall has a trust relationship with a set of other firewalls, and trusts worm alerts from its friends. Each firewall makes its own decision about whether a worm attack is in progress by a formula combining the number of friends alerting it and its own observations. This is then used to set up appropriate filters. Senthilkumar et al [51] extend this work to construct hierarchical architectures where a parent firewall is alerted by its children, and sends such alerts to its parent. Security against malicious participants is enforced in [15, 51] by assuming the existence of a friend network or a trust relationship between parents and children. In contrast, we employ verification mechanisms that allow a firewall's alert to be verified. This allows a single firewall to potentially

get alerted by any other firewall in a secure fashion (not just its set of friends or its parent). Anagnostakis et al [14, 52] propose a signaling protocol among participating firewalls to detect and contain a worm. This work attempts to monitor multiple worms at the same time, and is not suitable for very fast worms (as observed by the authors themselves). Our scheme generalizes on these two schemes in two ways. First, they employ only explicit signaling to propagate information. Determining the rate at which explicit signals should be propagated depends on the worm itself. Implicit signaling, as we will show later, does not suffer from this limitation. Secondly, the degree of resilience of these schemes against malicious participants has not been fully characterized. They mostly consider only inaccurate information from participants. We qualitatively analyze our scheme for its resilience and its containment as well.

8 Conclusion

In this paper, we have studied cooperative schemes among mutually untrusted firewalls in order to detect and contain unknown fast scanning worms. We identify three important components of our architecture: local detection, propagation, and filtering. We discuss two types of propagation: implicit signaling and explicit signaling. Under this framework, we derive analytical results for the effectiveness of each of these mechanisms. Detection and filtering are easy to implement and are effective under certain regimes in the complete deployment scenario, but would fail against some recent worms that have faster scanning rate and higher vulnerable populations. When implicit signaling is also used, the effectiveness improves, and it can offer containment of over 97% against most scanning worms. Implicit signaling has two main advantages: no additional overhead is required for signaling and its performance is independent of the scanning rate of the worm. Explicit signaling can be used to augment implicit signaling, and improves the containment metric over 99%. Explicit signaling is more resilient to smart worms and malicious firewalls that attempt to subvert our scheme. We also designed a rerouting scheme to deal with partial deployment: with 1% deployment, we show that our schemes can still provide containment (among deployed firewalls) of over 97%. We also extend our rerouting mechanism to handle malicious firewalls: this can handle a few malicious firewalls (of the order of a few hundred) in our cooperative.

Our analysis of these schemes is necessarily simplistic, and only offers analytical support for the performance of our schemes. While this is useful for deriving bounds on the effectiveness of cooperation, it does not address several vagaries of the Internet, such as firewalls protecting networks of various sizes, non-uniform vulnerable host distribution, mobile laptops that can spread infection covertly between networks, and worm-induced network congestion. As part of future work, we are interested in analyzing the effectiveness of our schemes under more realistic scenarios. We are currently in the process of implementing a prototype which would also include several overhead optimizations. For example, a firewall could implement a hybrid signaling protocol by using explicit signaling only when there are no outgoing packets to piggyback its implicit signals. The overhead of explicit signaling can be reduced by some form of secure application-level multicast. Other possible areas of future work include the design of local detection schemes to deal with other classes of worms and to incorporate them in our framework.

References

[1] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver, "Inside the Slammer Worm," *IEEE Security and Privacy*, vol. 1, no. 4, pp. 33–39, 2003.

[2] Colleen Shannon and David Moore, "The spread of the witty worm," Mar 2004, `http://www.caida.org/analysis/security/witty/`.

[3] David Moore, Colleen Shannon, and Jeffery Brown, "Code-red: a case study on the spread and victims of an internet worm," in *Proceedings of the Internet Measurement Workshop (IMW)*, 2002.

[4] Cliff Changchun Zou, Weibo Gong, and Don Towsley, "Code red worm propagation modeling and analysis," in *Proceedings of the 9th ACM conference on Computer and communications security*, 2002, pp. 138–147.

[5] Jose Nazario, "The Blaster worm: The view from 10,000 feet," `http://monkey.org/~jose/presentations/blaster.d/`.

[6] CAIDA, " Dynamic Graphs of the Nimda worm," `http://www.caida.org/dynamic/analysis/security/nimda/`.

[7] Stuart Staniford, Vern Paxson, and Nicholas Weaver, "How to Own the Internet in Your Spare Time," in *Proceedings of the 11th USENIX Security Symposium*, 2002, pp. 149–167.

[8] Nicholas Weaver, Vern Paxson, Stuart Staniford, and Robert Cunningham, "A Taxonomy of Computer Worms," in *Proc. of the ACM workshop on Rapid Malcode*, 2003, pp. 11–18.

[9] David Moore, Colleen Shannon, Geoffrey M. Voelker, and Stefan Savage, "Internet quarantine: Requirements for containing self-propagating code," in *IEEE INFOCOM*, 2003.

[10] Nicholas Weaver, Dan Ellis, Stuart Staniford, and Vern Paxson, "Worms vs Perimeters: The Case for HardLANs," in *Proc. Hot Interconnects*, Aug 2004.

[11] Vinod Yegneswaran, Paul Barford, and Somesh Jha, "Global Intrusion Detection in the Domino Overlay System," in *Proceedings of Network and Distributed System Security Symposium (NDSS)*, Feb 2004.

[12] Nicholas Weaver, Vern Paxson, and Stuart Staniford, "Wormholes and a Honeyfarm: Automatically Detecting Novel Worms," in *DIMACS Large Scale Attacks Workshop presentation*, Piscataway, NJ, Sep 2003, `http://www.icir.org/vern/dimacs-large-attacks/weaver.ppt`.

[13] Nicholas Weaver, Stuart Staniford, and Vern Paxson, "Very Fast Containment of Scanning Worms," in *Proceedings of the 13th USENIX Security Symposium*, 2004, pp. 29–44.

[14] K. G. Anagnostakis, M. B. Greenwald, S. Ioannidis, A. D. Keromytis, and D. Li, "A Cooperative Immunization System for an Untrusting Internet," in *Proceedings of the 11th IEEE International Conference on Networks (ICON'03)*, Oct 2003.

[15] D. Nojiri, J. Rowe, and K. Levitt, "Cooperative Response Strategies for Large Scale Attack Mitigation," in *Proceedings of The Third DARPA Information Survivability Conference and Exposition (DISCEX III)*, Washington, DC, April 2003.

[16] H. A. Kim and B. Karp, "Autograph: Toward automated, distributed worm signature detection," in *Proceedings of the 13th Usenix Security Symposium*, August 2004.

[17] "Route Views ," `http://www.routeviews.org/`.

[18] Stuart E. Schechter, Jaeyeon Jung, and Arthur W. Berger, "Very Fast Containment of Scanning Worms," in *The Seventh International Symposium on Recent Advances in Intrusion Detection (RAID)*, Sep 2004.

[19] Jaeyeon Jung, Vern Paxson, Arthur W. Berger, and Hari Balakrishnan, "Fast portscan detection using sequential hypothesis testing," in *Proc. of the IEEE Symposium on Security and Privacy*, May 2004.

[20] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage, "Automatic Worm Fingerprinting," in *Proceedings of OSDI*, San Francisco, USA, December 2004.

[21] N. Weaver, I. Hamadeh, G. Kesidis, and V. Paxson, "Preliminary results using scaledown to explore worm dynamics," in *Proceedings of ACM CCS WORM*, Washington, D.C., Oct 2004.

[22] Sheldon M. Ross, *Introduction to Probability Models, 8th Edition*, Academic Press, 2003.

[23] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," in *Proc. ACM SIGCOMM Conference (SIGCOMM '02)*, August 2002, pp. 73–78.

[24] Oleg Kolesnikov, Dave Dagon, and Wenke Lee, "Advanced Polymorphic Worms: Evading IDS by Blending in with Normal Traffic," 2004, `http://www.cc.gatech.edu/~ok/w/ok_pw.pdf`.

[25] M. Roesch, "Snort: Lightweight intrusion detection for networks," in *Proceedings of the 13th Systems Administration Conference*, 1999.

[26] Vern Paxson, "Bro: a system for detecting network intruders in real-time," *Computer Networks (Amsterdam, Netherlands: 1999)*, vol. 31, no. 23–24, pp. 2435–2463, 1999.

[27] Crispan Cowan, Calton Pu, Dave Maier, Jonathan Walpole, Peat Bakke, Steve Beattie, Aaron Grier, Perry Wagle, Qian Zhang, and Heather Hinton, "StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks," in *Proc. 7th USENIX Security Conference*, San Antonio, Texas, Jan 1998, pp. 63–78.

[28] Christina Warrender, Stephanie Forrest, and Barak A. Pearlmutter, "Detecting intrusions using system calls: Alternative data models," in *IEEE Symposium on Security and Privacy*, 1999, pp. 133–145.

[29] Manuel Costa, Jon Crowcroft, Miguel Castro, Antony Rowstron, Colleen Shannon, and Jeffery Brown, "Can we contain Internet worms?," in *Third Workshop on Hot Topics in Networks*, San Diego, USA, Nov 2004.

[30] J. Twycross and M. M. Williamson, "Implementing and testing a virus throttle," in *Proceedings of the 12th USENIX Security Symposium*, Aug 2003, pp. 285–294.

[31] Stuart Staniford, James A. Hoagland, and Joseph M. McAlerney, "Practical automated detection of stealthy portscans," *J. Comput. Secur.*, vol. 10, no. 1-2, pp. 105–136, 2002.

[32] S. Singh, C. Estan, G. Varghese, and S. Savage, "The EarlyBird System for Real-time Detection of Unknown Worms," 2003, Technical Report CS2003-0761, UCSD.

[33] Christian Kreibich and Jon Crowcroft, "Honeycomb: Creating intrusion detection signatures using honeypots," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, pp. 51–56, 2004.

[34] Niels Provos, "A Virtual Honeypot Framework," in *Proc. of 13th USENIX Security Symposium*, San Diego, CA, Aug 2004, pp. 51–56.

[35] David Moore, Colleen Shannon, Geoffrey M. Voelker, and Stefan Savage, "Network telescopes," Tech. Rep. TR-2004-04, CAIDA, 2003.

[36] "Counterpane Internet Security," `http://www.counterpane.com`.

[37] J. Wu, S. Vangala, L. Gao, , and K. Kwiat, "An Effective Architecture and Algorithm for Detecting Worms with Various Scan Techniques," in *Proc. of Network and Distributed System Security Symposium*, Feb 2004.

[38] Cliff Changchun Zou, Lixin Gao, Weibo Gong, and Don Towsley, "Monitoring and early warning for internet worms," in *Proceedings of the 10th ACM conference on Computer and communication security*, 2003, pp. 190–199.

[39] Brent N. Chun, Jason Lee, and Hakim Weatherspoon, "Netbait: a distributed worm detection service," Tech. Rep. IRB-TR-03-033, Intel Research Berkeley, Sep 2003.

[40] S. Staniford, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle, "GrIDS – A graph-based intrusion detection system for large networks," in *Proceedings of the 19th National Information Systems Security Conference*, 1996.

[41] "CounterMalice, Silicon Defense ," `http://www.silicondefense.com/products/countermalice/` (currently unavailable).

[42] S. Staniford, "Containment of scanning worms in enterprise networks," *Journal of Computer Security*, 2004 (to appear).

[43] K. Park, H. Kim, B. Bethala, and A. Selcuk, "Scalable Protection against DDoS and Worm Attacks," Tech. Rep., DARPA ATO FTN, Final Project Report, Purdue University, 2003.

[44] Cynthia Wong, Chenxi Wang, Dawn Song, Stan Bielski, and Gregory R. Ganger, "Dynamic Quarantine of Internet Worms," in *Proceedings of International Conference on Dependable Systems and Networks (DSN)*, Florence, Italy, June 2004.

[45] Helen J. Wang, Chuanxiong Guo, Daniel R. Simon, and Alf Zugenmaier, "Shield: vulnerability-driven network filters for preventing known vulnerability exploits," in *Proceedings of SIGCOMM*, 2004, pp. 193–204.

[46] S. Sidiroglou and A. Keromytis, "A network worm vaccine architecture," in *Proceedings of the IEEE Workshop on Enterprise Technologies: Infrastructure for Collaborative Enterprises (WETICE), Workshop on Enterprise Security*, Jun 2003.

[47] S. Sidiroglou and A. Keromytis, "Countering network worms through Automatic Patch Generation," Nov 2003, Columbia University technical report CUCS-029-03. New York, NY.

[48] V. Berk, G. Bakos, and R. Morris, "Designing a framework for active worm detection on global networks," in *IEEE International Workshop on Information Assurance*, Darmstadt, Germany, Mar 2003.

[49] Xuan Chen and John Heidemann, "Detecting early worm propagation through packet matching," Tech. Rep. ISI-TR-2004-585, USC/Information Sciences Institute, Feb 2004.

[50] Joel Sandin, "P2P Systems for Worm Detection," in *DIMACS Large Scale Attacks Workshop presentation*, Piscataway, NJ, Sep 2003, `http://www.icir.org/vern/dimacs-large-attacks/sandin.ppt`.

[51] C. G. Senthilkumar and K. Levitt, "Hierarchically Controlled Co-operative Response Strategies for Internet Scale Attacks," in *Proceedings of The International Conference on Dependable Systems and Networks*, San Francisco, CA, June 2003.

[52] Kostas G. Anagnostakis, Michael B. Greenwald, and Angelos D. Keromytis, "Efficiently detecting and reacting to day-zero worms on the internet," `http://citeseer.ist.psu.edu/634900.html`.

A Partial Deployment: Modeling Propagation

We now extend the model in Section 4.3 to allow for partial deployment. We analyze our two solutions under partial deployment: first, without rerouting, and second, with rerouting enabled. Unfortunately, we could not obtain any closed-form expressions in this scenario, and hence present a differential equation model along with numerical results. We consider the case when both implicit signaling and explicit signaling at the rate of E are used.

When there is no rerouting, the undeployed firewalls do not participate in detection or propagation. We use N, n, m to denote the total number of vulnerable firewalls, the number of deployed vulnerable firewalls and the number of undeployed vulnerable firewalls respectively ($N = (m + n)$). Denote by $m_i(t)$, the number of undeployed firewalls infected by time t. The following equations hold:

$$\begin{aligned}
\frac{d}{dt}(m_i) &= \frac{phs(m_i + n_i - n_d)(m - m_i)}{N} \\
&= s_n(m_i + n_i - n_d)(m - m_i) && (13)\\
\frac{d}{dt}(n_i) &= \frac{phs(m_i + n_i - n_d)(n - n_i - n_a)}{N} \\
&= s_n(m_i + n_i - n_d)(n - n_i - n_a) && (14)\\
\frac{d}{dt}(n_d) &= \frac{n_i - n_d}{t_d} && (15)\\
\frac{d}{dt}(n_a) &= \frac{hs(n_d)(n - n_i - n_a)}{N} + \frac{En_d(n - n_i - n_a)}{n} \\
&= \frac{s_n(n_d)(n - n_i - n_a)}{p}\left(1 + \frac{E}{hs\alpha}\right) && (16)
\end{aligned}$$

Equation (13) tracks the growth of infected undeployed firewalls over time (all $m - m_i$ uninfected firewalls are vulnerable), while Equation (14) simply replaces the term $n_i - n_d$ in Equation (3) with $m_i + n_i - n_d$. Note that in Equation (16), the first term accounts for implicit signaling, while the second term includes the explicit signaling term. Note that implicit signals are piggybacked on scans and are sent to all firewalls, while explicit signals are sent only to deployed firewalls.

When rerouting is enabled, we argue that the undeployed firewalls also behave as deployed firewalls with $t'_d = t_d/\alpha$ where $\alpha = n/N$ (the fraction of deployed firewalls). The argument is that the traffic seen by the firewall $A(X)$ is a fraction α of the traffic sent by X, and so is the number of failed connections. Since our local detection scheme uses a count of connection failures, $A(X)$ detects the infection of X by the fraction $1/\alpha$ slower than X itself would have done if it were deployed. Thus, the above analysis can be tweaked to accommodate this fact (we use $m_d(t)$ to denote the number of undeployed firewalls that have been identified as infected by their corresponding analysis firewalls at time t):

$$\begin{aligned}
\frac{d}{dt}(m_i) &= \frac{phs(m_i + n_i - n_d)(m - m_i)}{N} \\
&= s_n(m_i + n_i - n_d)(m - m_i) && (17)\\
\frac{d}{dt}(m_d) &= \frac{\alpha(m_i - m_d)}{t_d} && (18)\\
\frac{d}{dt}(n_i) &= \frac{phs(m_i - m_d + n_i - n_d)(n - n_i - n_a)}{N} \\
&= s_n(m_i - m_d + n_i - n_d)(n - n_i - n_a) && (19)\\
\frac{d}{dt}(n_d) &= \frac{n_i - n_d}{t_d} && (20)\\
\frac{d}{dt}(n_a) &= \frac{hs(m_d + n_d)(n - n_i - n_a)}{N} \\
&+ \frac{E(n_d + m_d)(n - n_i - n_a)}{n} \\
&= \frac{s_n(m_d + n_d)(n - n_i - n_a)}{p}\left(1 + \frac{E}{hs\alpha}\right) && (21)
\end{aligned}$$

There are two main differences from the equations for the previous case. Equation (19) uses the term $(m_i - m_d + n_i - n_d)$ for the number of scanning firewalls, since an undeployed firewall that has been identified as infected by its analysis firewall cannot send any more scans to a deployed network. Equation (21) also accounts from implicit and explicit signals sent from the m_d analysis firewalls.