WormShield: Fast Worm Signature Generation with Distributed Fingerprint Aggregation

Min Cai, *Member*, *IEEE*, Kai Hwang, *Fellow*, *IEEE*, Jianping Pan, *Member*, *IEEE*, and Christos Papadopoulos, *Senior Member*, *IEEE*

Abstract—Fast and accurate generation of worm signatures is essential to contain zero-day worms at the Internet scale. Recent work has shown that signature generation can be automated by analyzing the repetition of worm substrings (that is, fingerprints) and their address dispersion. However, at the early stage of a worm outbreak, individual edge networks are often short of enough worm exploits for generating accurate signatures. This paper presents both theoretical and experimental results on a collaborative worm signature generation system (*WormShield*) that employs distributed fingerprint filtering and aggregation over multiple edge networks. By analyzing real-life Internet traces, we discovered that fingerprints in background traffic exhibit a Zipf-like distribution. Due to this property, a distributed fingerprint filtering reduces the amount of aggregation traffic significantly. WormShield monitors utilize a new *distributed aggregation tree* (DAT) to compute global fingerprint statistics in a scalable and load-balanced fashion. We simulated a spectrum of scanning worms including CodeRed and Slammer by using realistic Internet configurations of about 100,000 edge networks. On average, 256 collaborative monitors generate the signature of CodeRedI-v2 135 times faster than using the same number of isolated monitors. In addition to speed gains, we observed less than 100 false signatures out of 18.7-Gbyte Internet traces, yielding a very low false-positive rate. Each monitor only generates about 0.6 kilobit per second of aggregation traffic, which is 0.003 percent of the 18 megabits per second link traffic sniffed. These results demonstrate that the WormShield system offers distinct advantages in speed gains, signature accuracy, and scalability for large-scale worm containment.

Index Terms—Network security, Internet worms, signature generation, worm containment, traffic measurement, distributed aggregation tree, distributed hash table, cardinality counting.

1 Introduction

Large-scale worm outbreaks are considered as a major security threat to today's Internet [1], [24], [25]. Network worms exploit vulnerabilities in widely deployed homogeneous software to self-propagate quickly on the Internet [38]. Recent advances in port-scan detection demonstrate that victims infected by scanning worms could be detected and quarantined quickly at individual edge networks [39]. On the other hand, to contain worms over the entire Internet, simulation studies [26] show that signature-based filtering is about 10 times faster than address blacklisting.

Automatic signature generation is essential for signature-based filtering to contain zero-day worms [16], [32]. It often takes hours or even days for security experts to extract worm signatures manually [32]. However, the reaction time of efficient worm containment could be less than a few hours or even minutes [26]. For example, Code-RedII worms infected more than 359,000 computers on the Internet in less than 14 hours [25]. Slammer worms probed all four billion IPv4 Internet addresses for potential victims in less than 10 minutes [24].

During the spreading of a monomorphic worm, the invariant content substrings shared by worm exploits are often repeated frequently, and their associated source or destination IP addresses are also dispersed widely. Recent work using Autograph [16] and Earlybird [32] shows that the procedure of signature generation can be automated by analyzing the repetition of content substrings (that is, fingerprints) and their address dispersion. These systems generally distinguish a worm signature from legitimate traffic patterns by some detection thresholds. However, infection attempts of a worm are often scattered around the entire Internet with a low density at its early outbreak stage. Consequently, individual edge networks may not be able to accumulate enough worm samples for fast and accurate signature generation.

Apparently, the overall infection attempts observed by multiple edge networks are more distinguishable from legitimate traffic than those observed by a single edge network. The more worm traffic that we observe and aggregate, the better the chance that we can generate accurate worm signatures sooner. Autograph [16] uses this heuristic to speed up signature generation by sharing IP addresses of port scanners among distributed monitors. However, Autograph does not share the repetition count of a substring among the monitors. The maximum repetition of a worm signature observed by an Autograph monitor is determined by the number of worm infection attempts

- *M. Cai is with the Department of Computer Science, University of Southern California, 3740 McClintock Ave., EEB 205, Los Angeles, CA 90089-2562. E-mail: mincai@usc.edu.*
- *K. Hwang is with the Department of Electrical Engineering, University of Southern California, 3740 McClintock Ave., EEB 212, Los Angeles, CA 90089. E-mail: kaihwang@usc.edu.*
- *J. Pan is with the Department of Computer Science, University of Victoria, PO Box 3055, STN CSC, Victoria, BC, Canada V8W 3P6. E-mail: pan@uvic.ca.*
- *C. Papadopoulos is with the Department of Computer Science, Colorado State University, USC 228, 1873 Campus Delivery, Fort Collins, CO 80523-1873. E-mail: christos@cs.colostate.edu.*

Manuscript received 26 Dec. 2005; revised 6 July 2006; accepted 20 Nov. 2006; published online 5 Feb. 2007.
For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-0177-1205. Digital Object Identifier no. 10.1109/TDSC.2007.1000.

targeting at the monitored edge network. Therefore, each Autograph monitor only observes the *local* rather than the *global* signature repetition.

Aggregating toward the global repetition and address dispersion of content substrings is challenging due to the requirement of scalability, fault tolerance, and load balance. The scalability is threefold. First, with increasing link speed, the number of payload substrings processed by each monitor will be tremendous even in a short time period, whereas the communication cost of global aggregation has to be moderate. Second, the aggregation has to scale up to a large number of monitors at multiple edge networks, for example, several thousands, if 10 percent of all edge networks are monitored. Third, the total number of distinct addresses of a suspicious substring observed by distributed monitors will be significantly large during a worm outbreak. Thus, the counting of global address dispersion must scale up to large sets of IP addresses with moderate communication cost.

Moreover, the aggregation has to be done in a fault-tolerant and load-balanced manner. Obviously, collecting all information at a central site will introduce a single point of failure, as well as communication and processing bottlenecks. Besides these technical challenges, global aggregation must preserve the privacy of individual organizations, since both content and address information are privacy sensitive. Preserving privacy is also necessary for encouraging different organizations to collaborate in distributed worm signature generation.

To meet these challenges, we propose a distributed worm signature generation system called *WormShield*. In WormShield, distributed monitors collaboratively generate the signatures of monomorphic worms by using distributed fingerprint filtering and aggregation at multiple edge networks. Our preliminary reports on WormShield [3], [13] described the basic concept of collaborative worm containment over *distributed hash table* (DHT) overlays [35]. This paper makes five unique contributions beyond our previous work:

1. analytical modeling of global worm-spreading properties,
2. efficient fingerprint filtering due to Zipf-like distribution,
3. distributed aggregation trees (DATs) with load balancing,
4. distributed estimation of address dispersion using an adaptive counting algorithm, and
5. large-scale simulations on signature generation speed, false positives, and deployment scalability.

The remainder of this paper is organized as follows: We review related work in Section 2. In Section 3, we mathematically model three worm-spreading properties and give an overview of our WormShield system. Section 4 studies the effectiveness of distributed fingerprint filtering at each monitor. We then present our DAT construction algorithms in Section 5, and our approach to estimating global address dispersion in Section 6. Section 7 evaluates the performance of collaborative WormShield monitors. Section 8 offers further discussion, and Section 9 concludes the entire paper.

2 Background and Related Work

Recently, there have been many research efforts on network worms [38], such as worm modeling and simulation [42], [10], [26], [22], measurement [25], [24], [1], and defense [39], [32], [16], [37]. Most scanning worms find a vulnerable target by randomly looking through the IP address space. There are several proposals to detect worm attacks by analyzing their scanning activities, for example, threshold random walks [39], [14] at individual edge networks and "trend detection" [42] on the global Internet. Another worm-detection approach is to passively monitor the scanning traffic sent to an unused address space, for example, network telescopes [25], [24], honeypots [34], and active sinks in DOMINO [41].

Network-based worm containment techniques can be classified into two major categories, that is, *address blacklisting* and *signature-based filtering*. The former quarantines infected hosts that exhibit abnormal port-scan activities [39], which is an efficient approach for protecting individual edge networks. However, to contain worms over the entire Internet, Moore et al. [26] show that signature-based filtering [30], [28] is more efficient than address blacklisting. Besides network-based techniques, Vigilante [6] employs the collaboration among end hosts to contain worms by using self-certified alerts. Shield [36] installs host-based network filters that are vulnerability specific and exploit generic once a vulnerability is discovered and before a patch is applied.

Our work was inspired by previous efforts on automatic signature generation for unknown worms, for example, Autograph [16] and Earlybird [32]. These two systems both employ the heuristic that the invariant byte string in worm payload will repeat very frequently during a worm outbreak. Autograph uses the flow-level heuristic of a worm, for example, port scanning, to classify suspicious flows before generating signatures. Distributed monitors in Autograph share the address information of port scanners to speed up the signature generation process. In contrast, Earlybird does not rely on any classified traffic. It uses the address dispersion of a byte string as another heuristic to distinguish a worm signature from legitimate traffic patterns. Several scalable algorithms, for example, multistage filter and scaled bitmaps, are used in Earlybird for the wire-speed implementation of a single monitor.

Although the basic concept of sharing the information among monitors is similar in Autograph [16] and WormShield, the shared information and underlying communication mechanisms are quite different in these two systems. Autograph uses application-level multicast over DHTs to share source IP addresses of port scanners among monitors, whereas WormShield uses DATs to compute the global fingerprint repetition and address dispersion. Fingerprint aggregation in WormShield has a considerably higher scalability requirement than the sharing of the IP addresses of port scanners in Autograph. Application-level multicast will impose too much overhead for fingerprint aggregation, for example, $O(N^2)$ messages per aggregation for N monitors. In contrast, DAT only uses $O(N)$ messages per aggregation. Furthermore, WormShield leverages distributed fingerprint filtering to reduce aggregation traffic significantly due to the Zipf-like fingerprint distribution. In summary, our work on WormShield is complementary to Autograph [16] and Earlybird [32], since

TABLE 1
Comparison of Six Worm Signature Generation Systems

Systems	Autograph	Earlybird	Polygraph	PAYL	WormShield	Vigilante
Deployment location	Network	Network	Network	Network	Network	Host
Information shared among monitors	Port-scan alerts	None	None	Z-Strings	Fingerprint statistics	Self-certified alerts
Information sharing method	Multicast	None	None	Centralized servers	Aggregation trees	Broadcast
Signature generation heuristics	Local prevalence of substrings	Local fingerprint repetition & address dispersion	Local prevalence of short string tokens	Mahalanobis distance of 1-gram	Global fingerprint repetition & address dispersion	Non-executable pages & dynamic dataflow analysis
Signature structure	Single substring	Single substring	Disjoint string tokens	Multiple substrings	Single substring	Self-certified alerts
Flow classification	Required	No	Required	Required	No	No
Polymorphic worms	No	No	Yes	No	No	Yes

distributed fingerprint filtering and aggregation can be used to improve the two systems as well.

PAYL [37] uses the "Z-string" of packet payload to generate worm signatures automatically. The payload alerts from different sites are correlated to increase accuracy and reduce false alarms. The privacy of individual sites is preserved by only exchanging unrecoverable Z-strings. Instead of generating worm signatures by using single substrings that could be evaded by polymorphic worms, Polygraph [27] generates the signatures of polymorphic worms with multiple disjoint string tokens that are shorter than the single substrings used in Autograph and Earlybird.

Collaborative intrusion detection, in general, and worm containment, in particular, have been studied in previous work. DOMINO [41] builds an overlay network among active-sink nodes to distribute alert information by hashing the source IP addresses. Worminator [23] summarizes port-scan alerts in Bloom filters and disseminates them among collaborating peers. Kannan et al. [15] analyze the efficacy of cooperation among firewalls in containing worms. In Vigilante [6], worms are collaboratively contained by distributing self-certified alerts among end hosts that do not necessarily trust each other. Our work is more focused on collaboratively generating worm signatures by using distributed fingerprint filtering and aggregation in multiple edge networks.

Table 1 compares the six worm signature generation systems according to seven criteria. WormShield is different from other systems in both its method of information sharing and the heuristics of signature generation. In WormShield, distributed monitors share fingerprint statistics with DATs, and worm signatures are generated by monitoring the *global* fingerprint repetition and address dispersion.

3 THE WORMSHIELD SYSTEM MODEL

In this section, we first analyze three important worm properties during its spreading. Then, we briefly describe the basic scheme of WormShield and its building blocks.

3.1 Modeling Worm-Spreading Properties

Worm outbreaks often exhibit some unique properties that are deviated significantly from legitimate traffic. We assume that all infection sessions of a worm share at least an invariant substring, that is, the worm signature. This assumption is valid for most existing monomorphic worms in the wild. We will discuss the limitation for polymorphic worms in Section 8.4. In the following, we analytically model three global worm-spreading properties during a worm outbreak, that is, *fingerprint repetition*, *dispersion of source address*, and *dispersion of destination address*. For the simplicity of our analysis, we assume a random scanning worm whose attack vector is always consistent. Also, we only consider the occurrence of fingerprints caused by a given worm instead of other benign traffic patterns.

A *fingerprint* is the Rabin-Karp [29] hash value of a content substring. In this paper, we use "fingerprint" and "substring" exchangeably, since the hash collision is ignorable when 64-bit fingerprint values are used. A *fingerprint repetition* is the repetition count of a given fingerprint. Let $I(t)$ be the number of infected hosts at time t, α be the worm probing rate, M be the vulnerable population, and I_0 be the initially infected population, where $1 \leq I_0 < M$. The *susceptible and infected (SI)* model [26], [42] has $I(t) = M\frac{e^{\beta(t-T)}}{1+e^{\beta(t-T)}}$, where $\beta = \alpha\frac{M}{2^{32}}$ for a 32-bit IPv4 address space, and T is an integration constant determined by I_0, that is, $T = ln(\frac{M}{I_0} - 1)/\beta$. Since $I(T) = \frac{M}{2}$, T represents the time to half of the vulnerable population being infected, let $r(t)$ be the *global fingerprint repetition* of a worm at time t. Since the worm fingerprint appears at least once in each worm exploit, we have

$$\begin{aligned} r(t) &= \int_0^t \alpha I(x)d(x) = \int_0^t \alpha M\frac{e^{\beta(x-T)}}{1+e^{\beta(x-T)}}d(x) \\ &= 2^{32}(ln(e^{\beta(t-T)}+1) - ln(e^{-\beta T}+1)). \end{aligned} \quad (1)$$

The *source address dispersion* of a fingerprint is defined by the number of distinct source addresses of the IP packets that contain the fingerprint. Let $s(t)$ be the *global source address dispersion* of a worm fingerprint at time t. Since each source address of the worm fingerprint represents an infected host, we have

$$s(t) = I(t) = M\frac{e^{\beta(t-T)}}{1+e^{\beta(t-T)}}. \tag{2}$$

Similarly, the *destination address dispersion* is defined by the number of distinct destination addresses of the IP packets that contain the fingerprint. Let $d(t)$ be the expected number of *global destination address dispersion* of a worm fingerprint at time t and $f(r)$ be the expected number of distinct destination IP addresses when a fingerprint repeats r times. Suppose a worm probes the 32-bit IPv4 space uniformly, and we have $f(r)$ distinct destination addresses after r probes. For the $(r+1)\text{th}$ probe, we have $f(r)+1$ distinct addresses with probability $(2^{32}-f(r))/2^{32}$ or $f(r)$ distinct addresses with probability $f(r)/2^{32}$. Therefore, we have

$$f(r+1) = (f(r)+1)(2^{32}-f(r))/2^{32} + f(r)f(r)/2^{32}.$$

Since $f(1)=1$, this equation can be solved, and $f(r) = 2^{32}(1-(1-1/2^{32})^r)$. Hence,

$$d(t) = f(r(t)) = 2^{32}(1-(1-1/2^{32})^{r(t)}). \tag{3}$$

According to (1), (2), and (3), both the fingerprint repetition and the address dispersion of a worm will increase exponentially at the early stage of its outbreak. When $r(t) \ll 2^{32}$, $d(t)$ increases almost linearly as a function of $r(t)$. These analytical models are further confirmed by the worm simulation results in Section 7.2. These three worm-spreading properties could be used to distinguish a worm signature from legitimate traffic patterns [16], [32].

3.2 Distributed Worm Signature Generation

Individual edge networks can only observe a small proportion of worm fingerprints. To have a better global view of worm activities, we propose a distributed worm signature generation system called WormShield. It consists of a set of geographically distributed monitors deployed at multiple edge networks or sites. All monitors organize themselves as a Chord overlay network [35]. Each monitor sniffs both inbound and outbound traffic on its access link to the backbone.

For monitor i in a network of n monitors, it first uses the Rabin-Karp algorithm [29] to compute the fingerprint of each sliding window in sniffed packets. The fingerprints are then sampled using a window-sampling algorithm [31]. Within a given window of fingerprints, the window-sampling algorithm selects the one with a minimal value. For each sampled fingerprint j, each monitor i updates its local fingerprint repetition $r_i(j)$, as well as the source and destination address sets $\mathbf{S}_i(j)$ and $\mathbf{D}_i(j)$, where $i=1,2,\ldots,n$. Let $s_i(j)$ and $d_i(j)$ be the source and destination address dispersions of j, respectively. We compute $s_i(j)=|\mathbf{S}_i(j)|$ and $d_i(j)=|\mathbf{D}_i(j)|$. Once $r_i(j)$, $s_i(j)$, and $d_i(j)$ all exceed their local thresholds, denoted by L_r, L_s, and L_d, respectively, fingerprint j becomes a local suspicious one and, then, is subject to global aggregation.

Similar to Earlybird, all fingerprints are actually clustered by destination port and protocol, since worms typically target a particular service [32]. The repetition and address dispersion are calculated for each fingerprint j with unique destination port and protocol. This will not reduce the ability to track worm traffic, but can effectively exclude a large amount of repetitive substrings in nonworm traffic. In the remainder of this paper, we assume that a fingerprint is associated with a unique pair of destination port and protocol.

To aggregate the information globally, WormShield automatically selects a *root monitor* for each fingerprint j, denoted by $root(j)$, by using Chord *consistent hashing*, which is discussed in Section 5. The root monitor of j is responsible for calculating the global repetition and the source and destination address dispersions of j, denoted by $r_g(j)$, $s_g(j)$, and $d_g(j)$, respectively. By aggregating the updates from all monitors, we obtain the following expressions for a global setting:

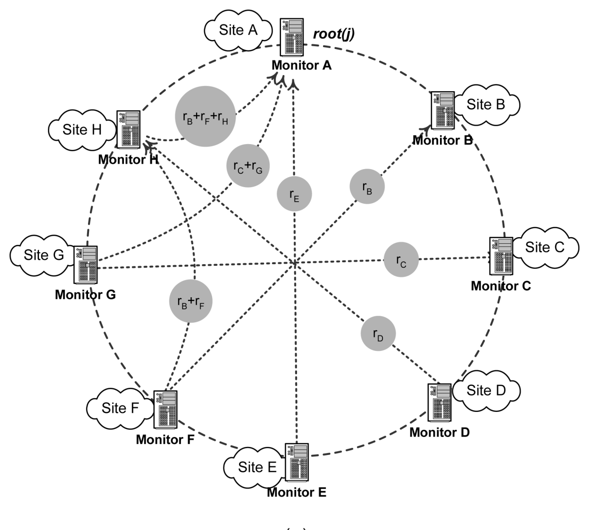
$$r_g(j) = \sum_{i=1, r_i(j)\ge L_r}^{N} r_i(j), \quad s_g(j) = \left|\bigcup_{i=1, s_i(j)\ge L_s}^{N} \mathbf{S}_i(j)\right|, \quad \text{and } d_g(j) = \left|\bigcup_{i=1, d_i(j)\ge L_d}^{N} \mathbf{D}_i(j)\right|. \tag{4}$$

Since the root monitor aggregates the information of the same fingerprint j from all monitors, it has a global view of the fingerprint repetition and address dispersion of j in all edge networks. If $r_g(j)$, $s_g(j)$, and $d_g(j)$ all exceed their global thresholds, denoted by G_r, G_s, and G_d, respectively, then the corresponding substring of j will be identified as a potential worm signature. Algorithm 1 sketches the basic process of distributed signature generation in WormShield.

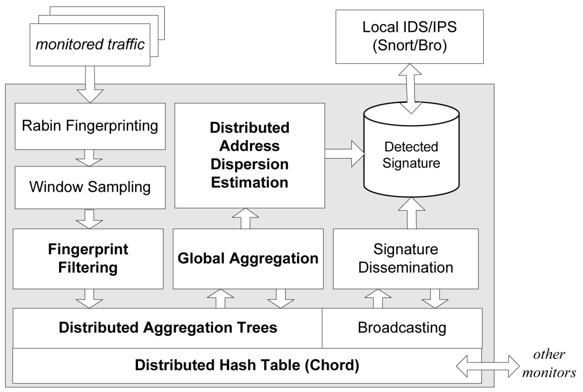
Algorithm 1. Distributed Signature Generation Algorithm

```
1:  INPUT: local thresholds($L_r$, $L_s$, and $L_d$) and global
      thresholds ($G_r$, $G_s$, and $G_d$)
2:  OUTPUT: generated worm signatures
3:  for all monitor $i$ in $1, 2, \ldots, n$ do
4:      for all local fingerprint $j$ in packets do
5:          $r_i(j) \leftarrow r_i(j) + 1$, $\mathbf{S}_i(j) \leftarrow \mathbf{S}_i(j) \cup \{\text{Src\_IP}(j)\}$,
            $\mathbf{D}_i(j) \leftarrow \mathbf{D}_i(j) \cup \{\text{Dest\_IP}(j)\}$
6:      end for
7:      if $r_i(j) \ge L_r$ AND $|\mathbf{S}_i(j)| \ge L_s$ AND $|\mathbf{D}_i(j)| \ge L_d$ then
8:          mark $j$ as a global fingerprint
9:          $r_g(j) \leftarrow r_g(j) + r_i(j)$, $\mathbf{S}_g(j) \leftarrow \mathbf{S}_g(j) \cup \mathbf{S}_i(j)$,
            $\mathbf{D}_g(j) \leftarrow \mathbf{D}_g(j) \cup \mathbf{D}_i(j)$
10:     end if
11: end for
12: for all global fingerprint $j$ at root monitor do
13:     if $r_g(j) \ge G_r$ AND $|\mathbf{S}_g(j)| \ge G_s$ AND $|\mathbf{D}_g(j)| \ge G_d$
        then
14:         output the substring of $j$ as a worm signature
15:     end if
16: end for
```

When monitors are uniformly distributed in the Chord identifier space, the number of fingerprints mapped to each monitor is almost uniformly distributed. However, if every monitor updates its local information to the root monitor directly using the Chord routing algorithm, then the root monitor of a potential worm signature will be overwhelmed by a large number of updates during a worm outbreak. Instead, WormShield constructs a DAT for each root monitor to aggregate the information gradually among all monitors rather than only at the root monitor. Each monitor



(a)



(b)

Fig. 1. Architecture of the WormShield system and functional design of its monitors. (a) Example of aggregating the fingerprint repetition in a WormShield network of eight nodes. (b) Main components of a WormShield monitor.

in the DAT will receive updates from its child monitors and send the *single* aggregated value (that is, the sum) to its parent monitor. Fig. 1a illustrates the process of aggregating the fingerprint repetition in WormShield.

Once a potential worm signature is identified, its root monitor then constructs a multicast tree on top of the Chord overlay network and disseminates the signature to all other monitors participating in the WormShield system. Other monitors could automatically deploy the received worm signatures in their local signature-based intrusion-detection systems such as Snort [30] and Bro [28].

The main building blocks of a WormShield monitor are shown in Fig. 1b. Each monitor implements a Chord protocol stack, on top of which the DAT and broadcasting modules are built. The DATs aggregate the global information of suspicious substrings that pass the fingerprint filtering at each monitor. The global address dispersions are then estimated in a distributed manner by using an adaptive counting algorithm. The signature database stores the signatures generated by a root monitor or disseminated by others, which could be further imported into a local intrusion detection/prevention system (IDS/IPS). In Sections 4, 5, and 6, we will detail the design of three major components: distributed fingerprint filtering, DATs, and distributed estimation of address dispersion.

4 Distributed Fingerprint Filtering

As we discussed in Section 3, WormShield uses fingerprint filtering at distributed monitors to filter out most content substrings in legitimate background traffic. Only those suspicious ones above local thresholds are subject to global aggregation. The problem of fingerprint filtering is similar to that of generating worm signatures at a single monitor in Earlybird [32]. However, the local thresholds should be low enough for fast global signature generation.

The fingerprint filtering scheme has two phases, that is, repetition filtering and address dispersion filtering. In the first phase, a multistage filter [11] is used to select *frequent* fingerprints that have repeated at least L_r times. In the second phase, the address dispersion of each frequent fingerprint is tracked using an adaptive counting algorithm presented in Section 6. A fingerprint is *dispersed* if it has at least L_s distinct source addresses and L_d distinct destination addresses. The fingerprint filter only selects suspicious fingerprints that are both frequent and dispersed, as specified in Algorithm 2.

Algorithm 2. The Fingerprint Filtering Algorithm

```
1: INPUT: a set of fingerprints **F** and the local thresholds
   $L_r$, $L_s$, and $L_d$
2: OUTPUT: locally suspicious fingerprints that are both
   frequent and dispersed
3: initialize a multistage filter $mf$ and an address
   dispersion table $ad$
4: for all fingerprint $f$ in F do
5:     if $ad[f]! = \mathrm{NULL}$ then
6:         add $f.src\_ip$ and $f.dst\_ip$ into sets $ad[f].src\_ip$ and
           $ad[f].dst\_ip$, respectively
7:         if $|ad[f].src\_ip| \geq L_s$ AND $|ad[f].dst\_ip| \geq L_d$ then
8:             output $f$ as a locally suspicious fingerprint
9:         end if
10:    else
11:        frequent $\leftarrow$ true
12:        for all stage $s$ in $mf$ do
13:            $s[f].count \leftarrow s[f].count + 1$
14:            if $s[f].count < L_r$ then
15:                frequent $\leftarrow$ false
16:            end if
17:        end for
18:        if frequent then
19:            add an entry $ad[f]$ for $f$ in table $ad$
20:        end if
21:    end if
22: end for
```

To aggregate as much information as possible, local thresholds need to be low enough. On the other hand, the higher the local thresholds are, the less aggregation traffic

TABLE 2
Summary of Eight Internet Packet Traces Used in Our Experiments

	Date	Duration	Direction	Packets	Bytes	Fingerprints
2005-IN-30s	08/18/2005	30 sec	inbound	0.3M	184.8M	169.3M
2005-OUT-30s	08/18/2005	30 sec	outbound	0.8M	713.0M	669.1M
2005-IN-1m	08/18/2005	1 min	inbound	0.7M	547.2M	509.4M
2005-OUT-1m	08/18/2005	1 min	outbound	1.5M	1191.6M	1110.1M
2005-IN-10m	08/18/2005	10 min	inbound	7.6M	5969.8M	5445.9M
2005-OUT-10m	08/18/2005	10 min	outbound	15.3M	12859.1M	12022.1M
2006-IN-10m	06/22/2006	10 min	inbound	9.3M	4731.1M	4250.9M
2006-OUT-10m	06/22/2006	10 min	outbound	11.5M	6737.0M	6138.4M

will be introduced by each monitor. Apparently, the effectiveness of fingerprint filtering is determined by the fingerprint characteristics in the background traffic, which is essential to estimate the aggregation overhead for a given link speed. We examined two kinds of fingerprint distributions, that is, the *distribution of fingerprint repetition* $X(r)$ and the *distribution of address dispersion* $Y(s,d)$. The former is the number of fingerprints that repeat r times, and the latter is the number of fingerprints that have s distinct source IP addresses and d distinct destination IP addresses.

We analyzed eight traces from two OC-24 access links of a class-B network in August 2005 and June 2006. The traces include both inbound and outbound traffic in 30-sec, 1-minute, and 10-minute time intervals. Table 2 summarizes the characteristics of these eight traces.

4.1 Distribution of Fingerprint Repetition

It is challenging to estimate the exact repetition distribution of fingerprints due to their large quantity; for example, the 10-minute inbound and outbound traces collected in 2005 have more than 7.6 million and 157 million fingerprints after $1/64$ sampling, respectively. Indeed, estimating the repetition distribution is similar to estimating flow size distribution [18]. Kumar et al. [18] proposed a probabilistic algorithm that uses *Expectation-Maximization* (EM) to estimate the flow size distribution. In their scheme, each fingerprint is first hashed into an index for an array of counters, and the counter at this index is incremented by 1. Hash collisions might cause two or more fingerprints to increment the same index. An iterative EM algorithm is then used to estimate the actual distribution from the value of counters. The EM algorithm can estimate large-scale distributions accurately with only moderate-sized memory, for example, 512 Mbytes of memory for 256 million fingerprints.

To verify the accuracy of EM estimation for our purpose, we implemented the EM algorithm, as well as a hash table approach. The latter calculates the *exact* repetition distribution by indexing a counter for each fingerprint. However, it is only able to analyze small-sized traces due to memory limitation. We define the *rank* of a fingerprint as its position in descending order of repetition counts. Fig. 2a compares the EM algorithm with the hash table approach by plotting the fingerprint repetition against its rank in trace 2005-IN-30s. The fingerprints are computed on 40 bytes k-grams, and $1/64$ window sampling is used to ease the memory requirement of the hash table implementation. The rank distributions of the fingerprint repetition for both approaches are plotted in $\log$-$\log$ scale. The close fitting of these two curves demonstrates that the EM algorithm is very accurate in estimating the repetition distribution. Indeed, the *weighted mean relative difference* (WMRD) [18] between the EM estimated distribution and the actual distribution is less than 0.01 percent.

Fig. 2a shows that window sampling retains the unbiased distribution for both low and high-repetitive fingerprints. The most frequent fingerprint in trace 2005-IN-30s repeats 3,199,055 times when no sampling is used. With $1/64$ window sampling, it is estimated to repeat 50,403 times, which is almost $1/64$ of the actual repetition. The $\log$-$\log$ scale plots in Fig. 2a reflect a linear relationship, which suggests that fingerprint repetition exhibits a Zipf-like distribution in small-sized traces. For large-sized traces collected in August 2005 and June 2006, Figs. 2b and 2c show a similar linear relationship between fingerprint repetition and its rank, which confirms that fingerprint repetition follows a Zipf-like distribution in both short and long time periods.

Zipf-like distributions are commonly observed in many kinds of phenomena although its exact cause is still unclear [20], [8]. The word "frequency" in randomly generated texts has a Zipf-like distribution, as observed in many nature languages such as English [20]. Also, fingerprints (that is, k-gram) in a broad class of unbiased binary texts exhibit Zipf-like distributions due to the presence of long-range correlation [8]. For example, fingerprints in Web documents follow a Zipf distribution [31]. Since a large portion of network traffic consists of Web documents, e-mail, or other contents, we believe that the fingerprints in a network traffic should exhibit a Zipf-like distribution as well.

In addition to the overall inbound and outbound traffic, we also analyzed the individual fingerprint distributions of the top three protocols with the highest traffic volumes. Fig. 2d plots the distributions of all top three protocols (FTP-data, NNTP, and HTTP) whose volumes are 82.7 percent of the total traffic in 2005-IN-10m. We observe that the fingerprints in each individual protocol also exhibit a Zipf-like distribution. Therefore, the Zipf-like distribution should be reliably seen in the overall traffic mixed with different protocols. The same results were also observed on other traces (2006-IN-10m and 2006-OUT-10m) collected almost one year later at a different access link.

4.2 Distribution of Address Dispersion

Besides the local repetition threshold, each WormShield monitor also applies local thresholds for the address dispersion of fingerprints. Therefore, we are also interested in the distribution of address dispersion $Y(s,d)$ in the

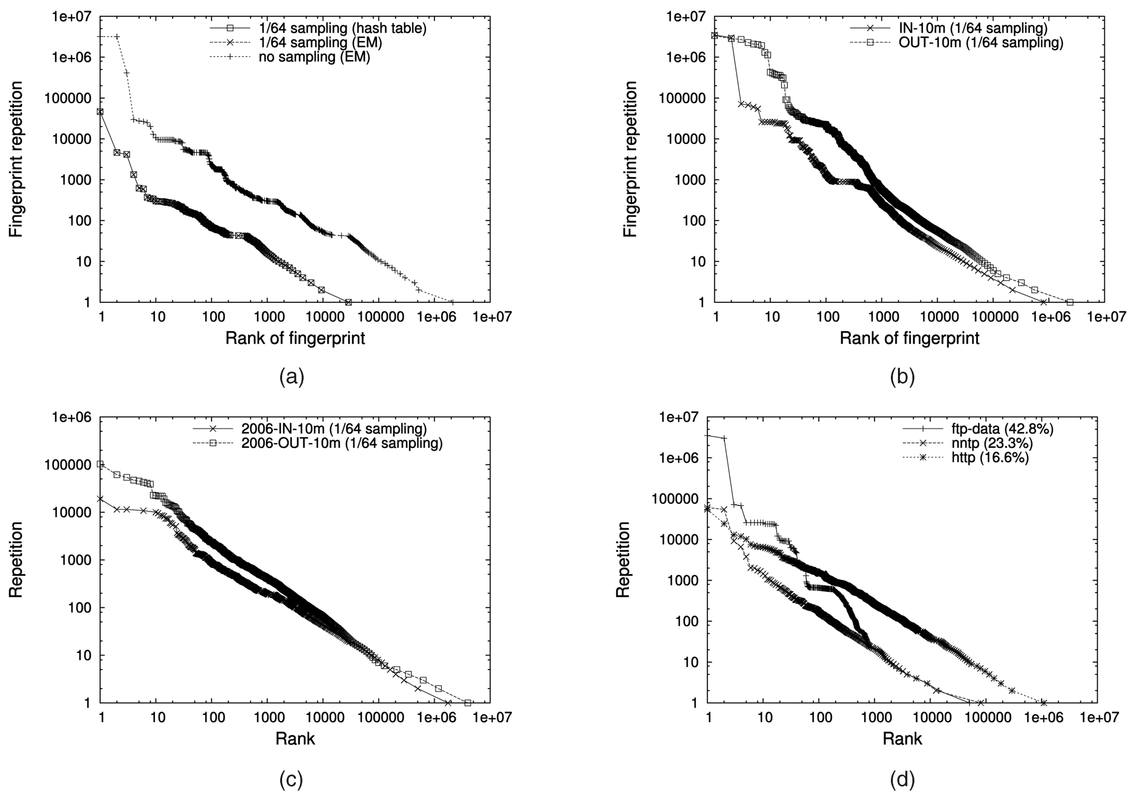


Fig. 2. The Zipf-like distribution of fingerprint repetition in various Internet traces. (a) Comparison of the hash table and the EM algorithm: inbound 30-sec trace (2005-IN-30s). (b) Inbound and outbound 10-minute traces on 18 August, 2005 (2005-IN-10m and 2005-OUT-10m). (c) Inbound and outbound 10-minute traces on 22 June, 2006 (2006-IN-10m and 2006-OUT-10m). (d) Top three protocols, with the highest traffic volume in trace 2005-IN-10m.

background traffic, where s and d are the source and destination address dispersions, respectively. We first apply a multistage filter to filter out most fingerprints that repeat less than 10 times in our experiments. Then, we estimate the address dispersion of fingerprints by using an adaptive counting algorithm discussed in Section 6. Fig. 3 shows the 3D histograms of the address dispersion in traces 2005-IN-10m and 2005-OUT-10m. Since the number of fingerprints is in $\log$ scale, most fingerprints only have a few distinct source and destination addresses.

4.3 Efficiency of Fingerprint Filtering

The Zipf nature of fingerprint repetition is critical to reduce the global aggregation overhead with fingerprint filtering in WormShield. We define the *filtering ratio* $F(t)$ as the fraction of fingerprints that exceed a given threshold t. Obviously, $F(t)$ needs to be as small as possible even when t is quite low. Fig. 4a plots the filtering ratio as a function of the repetition threshold in traces 2005-IN-10m and 2005-OUT-10m. It shows that the filtering ratio decreases dramatically when the threshold increases from 1 to 10. For example, the filtering ratio of 2005-IN-10m decreases from 0.01 to 0.0009 when the threshold increases from 1 to 5 and further decreases to 0.0003 when t is 10. Similarly, that of 2005-OUT-10m decreases from 0.015 to 0.0007 when the threshold increases from 1 to 5. Therefore, only about 0.1 percent of the fingerprints are subject to global aggregation when the local repetition threshold is only 5, which will almost not affect the efficacy of the global fingerprint repetition.

Similar to the case of fingerprint repetition, we also examine the filtering ratio $F(t)$ of address dispersion, where t is a threshold for both source and destination addresses. Fig. 4b shows that the filtering ratio also decreases dramatically when t increases from 1 to 10. For example, when t is applied to both source and destination addresses in trace 2005-IN-10m, $F(t)$ decreases from 0.04 to 0.001 when t increases from 1 to 10. Indeed, the distribution of address dispersion also follows a Zipf-like distribution, since the filtering ratio is actually a complementary cumulative function of the address dispersion distribution.

Therefore, even when considerably low thresholds are used, the fingerprint filtering is able to reduce the aggregation traffic from a single monitor by five or six orders of magnitude. Note that the overall filtering ratio is the multiplication of two filtering ratios in the repetition and address dispersion filtering phases. For each fingerprint, we use 20 bytes for its SHA-1 hash value, 4 bytes for its repetition value, and 90 bytes each for summarizing its source and destination addresses. When all local thresholds are 10, we experience roughly 0.6 kilobits per second of aggregation traffic from each monitor, which is about 0.003 percent of the 18 megabits per second link traffic.

5 Distributed Aggregation Trees for Collaborative Monitors

As mentioned in Section 3, a root monitor will become overloaded if all other monitors update their local information to the root monitor directly. Since the root monitor only

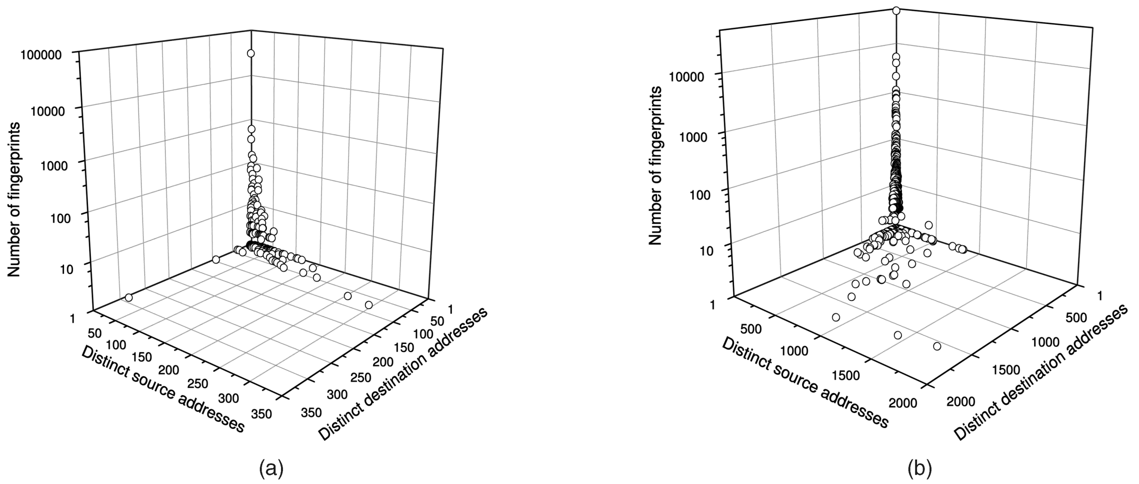


Fig. 3. The distribution of address dispersion in 3D histograms. (a) Inbound 10 minutes (2005-IN-10m). (b) Outbound 10 minutes (2005-OUT-10m).

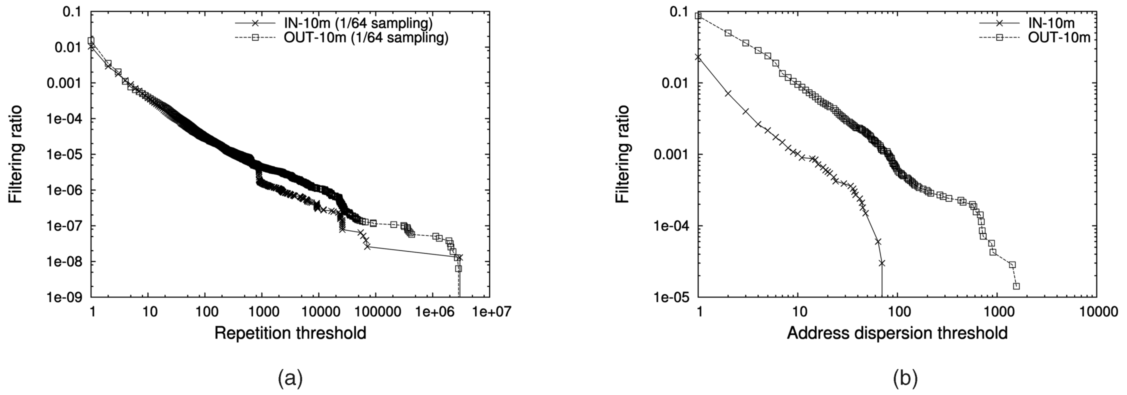


Fig. 4. The filtering ratio of packet traces decreases with increasing local thresholds. (a) Effects of the fingerprint repetition threshold. (b) Effects of the address dispersion threshold.

needs to collect the aggregated information, for example, the *sum* of repetition counts from all edge networks, aggregating the information gradually by all monitors will significantly reduce the communication and processing costs at root monitors. In this section, we illustrate how to build a *distributed aggregation tree* (DAT) implicitly for each root monitor on the Chord overlay network [35].

Our DAT construction is based on the Chord protocol [35]. Chord uses a circular b-bit identifier space with modulo 2^b for both node identifiers and object keys. Every node in Chord is assigned to a unique identifier by using a uniform hash function such as SHA-1. All nodes organize themselves into a ring topology according to their identifiers in the circular space. Object keys are assigned to nodes by using *consistent hashing*: key k is assigned to its *successor node*, that is, the first node whose identifier is equal to or follows k in the circular space, denoted by $successor(k)$.

Besides its immediate predecessor and successor nodes, every Chord node maintains a set of b finger nodes that are spaced exponentially in the identifier space. The jth finger of node i is the first node that succeeds i by at least 2^j in the identifier space, where $0 \leq j < b$. The finger table contains more nearby nodes than faraway nodes at a doubling distance. Chord uses *finger routing* to forward lookup messages. When node i wants to look up key k that is far away from i, it forwards a lookup message to the finger node whose identifier most immediately precedes $successor(k)$. By repeating this process, the message gets closer and closer to, and will eventually reach, $successor(k)$.

5.1 Basic DAT Construction

In WormShield, it is not feasible to build aggregation trees *explicitly* by maintaining the parent-child relationships, as suggested by Li et al. [19]. First, we need to build an aggregation tree for each root monitor, and an explicit tree construction cannot scale up to a large number of trees. Second, the parent-child maintenance overhead will be further exaggerated when some monitors join or leave the network. Even when only one monitor leaves the network, all trees need to be adjusted, since this monitor will be part of all aggregation trees.

We extend the basic Chord algorithm [35] to build a DAT rooted at each root monitor for aggregating toward the global repetition and address dispersion of a potential worm signature. Instead of building DATs explicitly, we employ Chord finger routing paths to *implicitly* build a DAT. Considering the root monitor r of a given fingerprint in an overlay network of n monitors, we need to construct a DAT $T(r)$ rooted at r. Suppose $\mathbf{P}$ is the set of paths from every node i to r using the Chord finger routing, where $i = 1, 2, \ldots, n$. $T(r)$ can be constructed using Algorithm 3.

Algorithm 3. DAT Construction Algorithm

```
1: INPUT: Chord routing paths $\mathbf{P}$ to root monitor $r$
```

```
2: OUTPUT: a DAT $T$ rooted at $r$
3: ROOT(T) $\leftarrow$ r
4: for all path $p$ in P do
5:     parent $\leftarrow$ ROOT(T)
6:     skip the last hop in path $p$
7:     for all node $i$ in path $p$ in reverse order do
8:         matched $\leftarrow$ false
9:         for all child $c$ of $parent$ do
10:            if $(c == i)$ then
11:                $parent \leftarrow c$
12:                matched $\leftarrow$ true
13:                break
14:            end if
15:        end for
16:        if not matched then
17:            CHILD(parent) $\leftarrow$ $i$
18:            append the rest path $p$ to $i$
19:            break
20:        end if
21:    end for
22: end for
```

Figs. 5a and 5b show an example of constructing a DAT rooted at node N_0 in a Chord network of eight nodes. In Fig. 5a, the label on each link shows the entry of the finger table used by Chord. For example, $N_i \rightarrow F[j]$ represents that the finger node, which is 2^j away from node N_i in the identifier space, is used for the Chord finger routing. This DAT construction algorithm can be easily extended to a distributed setting. Actually, distributed nodes do not need to build DATs explicitly. Instead, all the nodes know its parent directly by using the Chord finger routing; that is, the next hop in the forwarding path is the parent. Moreover, the resultant DAT can adapt to node arrival and departure quickly by using the Chord stabilization algorithm [35].

5.2 Balanced DAT Construction

Recall that Chord finger routing paths are at most $O(\log N)$ hops; therefore, the height of DAT will be at most $O(\log N)$. However, the branching factor is not the same for all nodes. The maximum branching factor is $O(\log N)$, since the root node has $O(\log N)$ child nodes. However, the minimum branching factor is 1 for the farthest node from the root. Thus, basic DATs are not balanced, and some nodes need to aggregate information from more child nodes than others. The imbalance is due to the greedy strategy in the Chord finger routing algorithm [35]. As we described before, a Chord node will always forward a message to the closest preceding node in its finger table. For example, node N_4 forwards its update to N_0 directly by using the finger 2^2 away in the identifier space from itself.

To build a DAT with a constant branching factor, we propose a *balanced routing* scheme for Chord. Instead of selecting a finger from the entire finger table, node i only considers a subset of fingers that are at most $2^{g(i,r)}$ away from i, where $g(i,r)$ is determined by the clockwise distance between i and r in the identifier space. We empirically determine $g(i,r) = \lceil \log \frac{(i-r) \mathrm{mod} 2^b + 2d}{3} \rceil$, where d is the average distance between two immediately adjacent nodes.

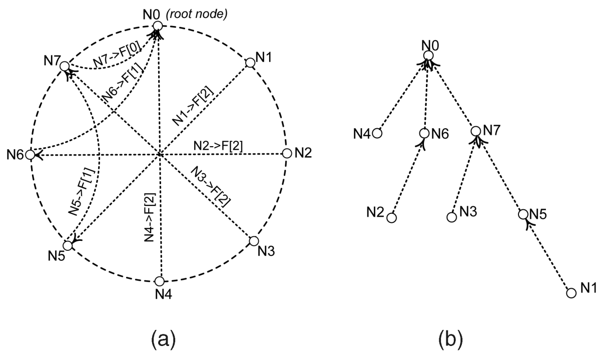


Fig. 5. Example of building a DAT by using finger routing paths in a Chord network of eight nodes. (a) Finger routing paths to N_0. (b) DAT rooted at N_0.

Figs. 6a and 6b demonstrate this balanced routing scheme and the more balanced DAT. Note that, in Fig. 6a, node N_4 only selects the closest preceding finger from the fingers that are at most 2^1 hops away from itself, since $g(4,0) = \lceil \log \frac{4 \mathrm{mod} 8 + 2}{3} \rceil = 1$. Therefore, node N_6 now is the next hop of N_4, whereas node N_0 was its next hop in Fig. 5a when the ordinary finger routing algorithm was used. The routing of all other nodes remains unchanged, and the DAT is more balanced, with a maximum branching factor of 2, as shown in Fig. 6b.

Since DATs are built implicitly without parent-child membership maintenance, they are scalable up to a large number of trees (for example, one per root monitor) and resilient to node dynamics as well. Aggregation in the network with DATs significantly reduces the imbalanced load at root monitors during a worm outbreak. Moreover, direct aggregation at root monitors requires $O(N \log N)$ essages for each aggregation from N monitors, since there are N updates, and each update will be routed in $O(\log N)$ hops by using the Chord finger routing. In contrast, aggregation with DAT requires minimal $O(N)$ messages over N monitors, since a parent node will aggregate updates from its child nodes and only send one update on their behalf to the parent node.

6 Distributed Estimation of Address Dispersion

As we defined in Section 3, the global address dispersion sets $\mathbf{S}_g(j)$ and $\mathbf{D}_g(j)$ of a given fingerprint j are the union of local address dispersion sets $\mathbf{S}_i(j)$ and $\mathbf{D}_i(j)$ from n monitors, where $i = 1, \ldots, n$. Instead of performing the union operation on all address sets directly, we indeed only need to estimate the number of *distinct* IP addresses in all sets, that is, the *cardinality* of the union of all address sets. Since the address sets may vary from empty to very large, we need to use a very small size memory to estimate cardinalities varying in several orders of magnitude.

6.1 Adaptive Cardinality Counting

The LogLog algorithm proposed by Durand and Flajolet [9] counts large cardinalities with high accuracy and little storage. For a set of elements, it first uses a uniform hash function to eliminate duplicates and generate uniformly distributed hash values. The hash values are separated into

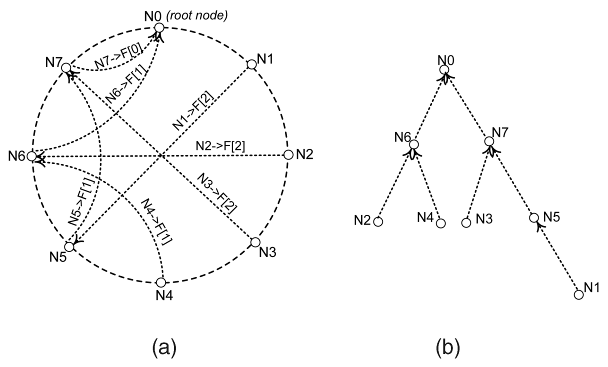


Fig. 6. Example of building a balanced DAT by using balanced routing paths in a Chord network of eight nodes. (a) Balanced routing paths to N_0. (b) Balanced DAT rooted at N_0.

m groups or *buckets*. For a hash value x in binary format, let $\rho(x)$ be the position of its first most significant bit 1. It uses m counters in memory, denoted by C_k, to record the largest ρ of hash values in bucket $k = 1, 2, \ldots, m$. The arithmetic mean $\sum_{k=1}^{m} C_k/m$ can be expected to approximate $\log_2(N/m)$ with an additive bias, where N is the number of distinct elements.

However, LogLog counting is asymptotically unbiased *only* when N is much greater than m. Since many buckets are empty when the load factor $t = N/m$ is relatively small, sampling error will introduce significant estimation bias. The probability that an element is hashed into a given bucket is $p = 1/m$. For a set of N distinct elements, the probability that a bucket is empty is

$$p_e = (1 - 1/m)^N \approx (1/e)^{N/m} = e^{-t},$$

that is, the number of empty buckets will increase exponentially as t decreases. Although LogLog counting is not suitable when t is relatively small, the expected number of empty buckets gives us a much better estimate of N. Suppose the number of empty buckets is b_e. The expected value of b_e after digesting N distinct elements is $\mathrm{E}[b_e] = mp_e \approx e^{-N/m}$, and N can be estimated as $-m\ln(b_e/m)$. This is similar to the *linear counting* algorithm that counts b_e by using a bitmap of m bits instead of m buckets [40].

Our finding is that there is a constant *switching load factor*, denoted by t_s, at which the standard error of linear counting becomes greater than that of LogLog counting. We calculate $t_s \approx 2.89$ analytically, which is independent of the number of buckets m. The *adaptive counting* algorithm uses the same data structure as LogLog counting, that is, an array of m counters, referred to as *cardinality summary* in our context. Suppose the ratio of empty buckets is β. We have $\beta_s = b_e/m = e^{-t_s} = 0.051$, where β_s is the ratio of empty buckets when $t = t_s$. Thus, N can be estimated as follows:

$$\hat{N} = \begin{cases} \alpha_m m 2^{\frac{1}{m}\sum_{k=1}^{m} C_k} & \text{if} \quad 0 \le \beta < 0.051 \\ -m\ln(\beta) & \text{if} \quad 0.051 \le \beta \le 1, \end{cases} \tag{5}$$

where α_m is a correction factor. In [4], it has been shown that the adaptive counting algorithm always gives better estimation even when the set cardinality changes over a wider range of magnitude.

6.2 Estimating Global Address Dispersion

With m b-bit counters, the adaptive counting can estimate up to N distinct addresses, and the relative error is bounded by $1.3/\sqrt{m}$, where $\log(\log(N/m) + 3) \le b$. For example, 144 counters with five bits each (a total of 90 bytes) can estimate up to 1.5×2^{32} distinct addresses, with a relative error less than 10.8 percent. Since there are only 2^{32} IPv4 addresses, a 90-byte cardinality summary is enough for estimating all possible address dispersions.

As we know, global address dispersion is quantified as the cardinality of the union of all address sets observed by individual monitors. Let $\overline{\mathbf{S}}_i(j)$ and $\overline{\mathbf{D}}_i(j)$ be the cardinality summaries of $\mathbf{S}_i(j)$ and $\mathbf{D}_i(j)$, respectively. To estimate the global address dispersions $|\mathbf{S}_g(j)|$ and $|\mathbf{D}_g(j)|$, we only need to merge the cardinality summaries $\overline{\mathbf{S}}_i(j)$ and $\overline{\mathbf{D}}_i(j)$ instead of the actual address sets $\mathbf{S}_i(j)$ and $\mathbf{D}_i(j)$ from all monitors. Then, $|\mathbf{S}_g(j)|$ and $|\mathbf{D}_g(j)|$ can be estimated from the merged cardinality summaries by using the adaptive counting algorithm.

Recall that the cardinality summary $\overline{\mathbf{S}}$ consists of an array of m counters, that is, $\overline{\mathbf{S}}[k] = C_k$, where $1 \le k \le m$, and C_k is the kth counter. We define the *max-merge* operator (denoted by $\circ$) as follows:

$$\overline{\mathbf{S}} = \overline{\mathbf{S}}_1 \circ \overline{\mathbf{S}}_2 \ \text{ iff } \ \forall k \in [1, m], \overline{\mathbf{S}}[k] = \max\{\overline{\mathbf{S}}_1[k], \overline{\mathbf{S}}_2[k]\}, \tag{6}$$

where $\overline{\mathbf{S}}$ is the max-merged cardinality summary of $\overline{\mathbf{S}}_1$ and $\overline{\mathbf{S}}_2$. This operator can be repeatedly applied to n cardinality summaries, denoted by $\overline{\mathbf{S}}_1 \circ \overline{\mathbf{S}}_2 \circ \ldots \circ \overline{\mathbf{S}}_n$. As we discussed in Section 6.1, the duplicate elements of a set are eliminated after applying the uniform hash function. Also, the cardinality summary of a set is independent of the processing sequence on its elements. Suppose $\mathbf{S}_g(j) = \bigcup_{i=1}^{n} \mathbf{S}_i(j)$. We have

$$\overline{\mathbf{S}}_g(j) = \overline{\mathbf{S}}_1(j) \circ \overline{\mathbf{S}}_2(j) \circ \ldots \circ \overline{\mathbf{S}}_n(j). \tag{7}$$

Therefore, $|\mathbf{S}_g(j)|$ can be estimated from the max-merged cardinality summary $\overline{\mathbf{S}}_g(j)$. Similarly, we can estimate $|\mathbf{D}_g(j)|$ from

$$\overline{\mathbf{D}}_g(j) = \overline{\mathbf{D}}_1(j) \circ \overline{\mathbf{D}}_2(j) \circ \ldots \circ \overline{\mathbf{D}}_n(j). \tag{8}$$

In WormShield, the max-merge operation is performed gradually through a DAT. Instead of collecting all cardinality summaries at a root monitor, each monitor max-merges the summaries from its child nodes and only sends the merged one to its parent in the DAT. Since cardinality summaries have $O(\log\log N)$ space complexity for N addresses, this scheme significantly reduces the communication cost of aggregating global address dispersion during a worm outbreak.

7 Simulation Results on Worm Signature Generation

In this section, we simulated a spectrum of scanning worms including CodeRed and Slammer on realistic Internet configurations. Although collaborative monitors are expected to perform better than isolated ones, we are interested in the quantitative comparison of WormShield and equal number of isolated monitors by various performance metrics, including signature generation speed, false positives, and deployment scalability.

7.1 Simulation Setup

To faithfully simulate worm propagation and signature generation events, we implemented a discrete event

TABLE 3
Parameters of Four Simulated Worms

Simulated worms	Date of BGP snapshot	# of total edge networks	Vulnerable population	# of vulnerable edge networks	Probing rate (probe/sec)	Probing method
CodeRedI-v2	July 19, 2001	97,461	359,000	59,004	10	uniform
CodeRedII	Aug. 4, 2001	97,876	359,000	59,343	10	subnet-preferred
Slammer-I	Jan. 25, 2003	112,065	75,000	40,136	4000	uniform
Slammer-II	Jan. 25, 2003	112,065	75,000	40,136	4000	subnet-preferred

simulator in packet level. It simulates three types of network objects, that is, vulnerable host, edge network that advertises a network address prefix, and *Autonomous System* (AS). To realistically partition the IP address space among edge networks, the simulator uses imported Border Gateway Protocol (BGP) snapshots at RouteViews (www.routeviews.org) to assign the IP address blocks of simulated edge networks and ASs [26], [16]. For each vulnerable edge network that contains at least one vulnerable host, a WormShield monitor is simulated to aggregate the fingerprint repetition and address dispersion of sniffed packets. The simulator was written with about 7,000 lines of C code and is available for download at http://gridsec.usc.edu/wormshield.

We simulated two variants of CodeRed and Slammer worms with both uniform and subnet-preferred scanning schemes. All worms have 25 hosts infected at the beginning of the simulation. Table 3 summarizes the parameters of these four simulated worms. The CodeRed and Slammer worms are used as two representative samples in a spectrum of TCP and UDP scanning worms. Note that these worms are simulated as *unknown* worms in our experiments. Neither WormShield nor isolated monitors take any advantage of their exploit code or associated vulnerabilities. Our simulation results are independent of the actual worm types used as long as they have similar propagation parameters, for example, vulnerable population, probing rate, and so on. We did not simulate the bandwidth limitation of access links, which may cause the Slammer worms to spread somewhat faster in our simulation than in a real network environment.

7.2 Spreading Patterns of Simulated Worms

To validate our simulator and the analytical model in Section 3.1, we simulated the spread of two Slammer worms and plotted their global fingerprint repetition and address dispersion over time, as shown in Fig. 7. The fingerprint repetition of Slammer-I with uniform scanning is well predicted in (1), as shown in Fig. 7a. Since Slammer-II uses the subnet-preferred scanning, its fingerprint repetition increases much faster than that of Slammer-I. We observe in Fig. 7b that the source address dispersions of both Slammer-I and Slammer-II are very close to the number of infected hosts, which is consistent with our analysis in (2). For uniform scanning worms (for example, Slammer-I), the destination address dispersion predicted in (3) also matches well with the simulation results, as shown in Fig. 7c.

Figs. 7a, 7b, and 7c show that both fingerprint repetition and address dispersion increase exponentially when the worm just starts spreading, which matches well with our theoretical analysis in Section 3.1. Our simulations on two CodeRed worms have similar results as well. Fig. 7d shows that when the destination address dispersion is far less than the IPv4 address space, it almost increases linearly with the fingerprint repetition, as illustrated in (3). Once the destination address dispersion of a worm signature exceeds a given threshold, its fingerprint repetition must be greater than the threshold as well. Therefore, signature generation systems only need to consider the global source and destination address dispersions as the criteria to identify a worm signature.

7.3 Speed of Signature Generation

Next, we compare collaborative WormShield monitors with isolated ones in terms of how timely they generate worm signatures. With different probing rates of worms, the absolute time duration may not be a good metric to measure the timeliness of signature generation. Instead, we characterize *the speed of signature generation* reversely by the number of hosts infected when a new worm signature is generated. We are particularly interested in evaluating the speed of signature generation under various thresholds.

We first simulate the signature generation of CodeRedI-v2 and CodeRedII with default 10 probes per second. For the sake of fair comparison, we deploy an equal number of isolated and WormShield monitors (that is, 256 monitors) in randomly selected edge networks. The isolated monitors do not share fingerprint statistics among themselves and generate a worm signature only based on their local information. Fig. 8 shows the number of infected hosts at the signature generation time as a function of different global thresholds for CodeRedI-v2 and CodeRedII. When only the source address dispersion is considered for CodeRedI-v2, as shown in Fig. 8a, the average case of isolated monitors succeeds in generating the signature before 343,400 out of 359,000 vulnerable hosts are infected when the global threshold is 100. The 99th percentile case almost generates the signature as slow as the average case, that is, 332,600 infected hosts. Since WormShield aggregates the global source address dispersion from all monitors, it generates the signature before 7,200 hosts are infected, which is roughly 1/48 of using isolated monitors.

When only the destination address dispersion is considered in Fig. 8b, the 99th percentile generates the signature much faster than the average case, for example, 29,500 and 307,400 infected hosts, respectively, when the global threshold is 3,000. This is because some isolated monitors are able to observe more outbound infection attempts if their edge networks have already had some infected hosts. WormShield monitors further reduce the number of infected hosts to 2,880 by aggregating the global destination addresses. Note that the 99th percentile case cannot always guarantee to achieve this, since there might not be an isolated monitor deployed in that edge network.

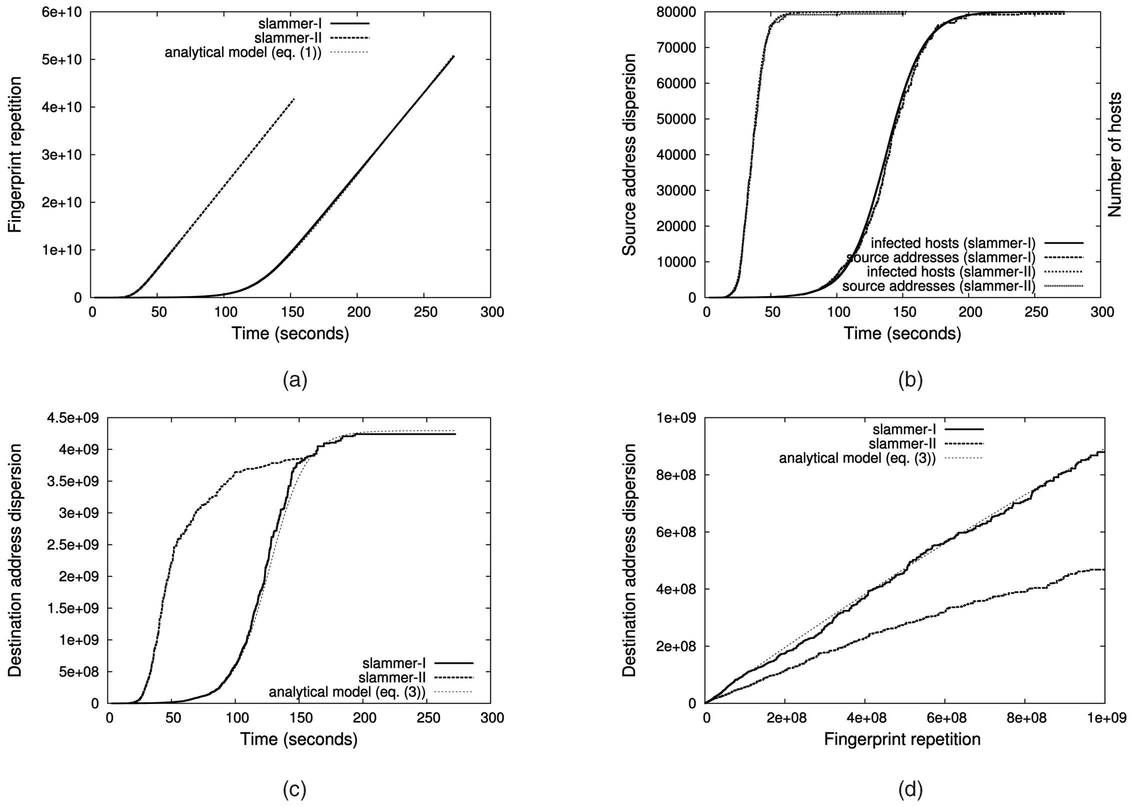


Fig. 7. The spreading patterns of Slammer worms with uniform and subnet-preferred scanning. (a) Fingerprint repetition over time. (b) Source address dispersion over time. (c) Destination address dispersion over time. (d) Destination address dispersion versus fingerprint repetition.

Figs. 8c and 8d show that WormShield monitors also generate the signature of subnet-preferred worms (that is, CodeRed-II) much faster than isolated monitors in both the average and the 99th percentile cases. In addition, neither isolated nor WormShield monitors are able to generate the signature as fast as that of CodeRedI-v2. This is because CodeRedII scans the IP addresses in its own subnet more frequently, whereas other addresses scan less frequently than CodeRedI-v2. However, both isolated and WormShield monitors only sniff inbound and outbound traffic on the access links of edge networks. They are not aware of the infection attempts inside the same edge network.

In Fig. 8, we observe that the global threshold of the destination address could be set much higher than that of the source address. Since the probing rate of a worm can be easily adjusted, we also study the effect of probing rate on the speed of signature generation. Fig. 9 plots the number of infected hosts for CodeRedI-v2 by varying the probing rate from 10 to 1,000. The global thresholds of the source and destination addresses are fixed at 30 and 3,000, respectively. Fig. 9 shows that the number of infected hosts decreases considerably when the probing rate increases from 10 to 200 for all three cases of using isolated and WormShield monitors. As the probing rate increases further from 200 to 1,000, there is no significant decrease in the number of infected hosts.

7.4 Effects of False Signatures

Signature accuracy is often quantified by two metrics, that is, *false negatives* and *false positives*. The evaluation of false negatives requires time-synchronized worm-attack traces at different edge networks. However, there is no such trace data publicly available in the research community. Instead, we simulated a worm attack synthesized with background traffic. In this case, WormShield and isolated monitors either succeed or fail to generate the signature of the simulated worm. Thus, false negatives cannot be meaningfully evaluated in our simulations.

We evaluate false positives with the number of *false signatures*, which are not real worm signatures. To understand the trade-off between signature generation speed and false signatures, we must conduct the worm simulation with realistic background traffic. We split the 10-minute inbound and outbound traces (2005-IN-10m and 2005-OUT-10m) of a class-B network into 256 traces of class-C networks. The partitioned traces capture most of the traffic at their access-links and only miss the traffic among those class-C networks.

Since the traces only last 10 minutes, we have to adjust the probing rate to fit the worm propagation in this time scale. The probing rate of two CodeRed worms is 200 probes per second in this trace-driven simulation. Fig. 10 shows that the number of false signatures drops sharply with increasing thresholds, for example, less than 10 false signatures when the global threshold is greater than 1,000. These false signatures are out of the 5.9-Gbyte inbound and 12.8-Gbyte outbound background traffic. The false-positive rate will be very low if we count the total number of fingerprints as the denominator. We note that there might be some worm noise in the background traffic although there is no known worm outbreak during the trace-collection time. Therefore, the number of actual false signatures might be even lower than that shown in Fig. 10.

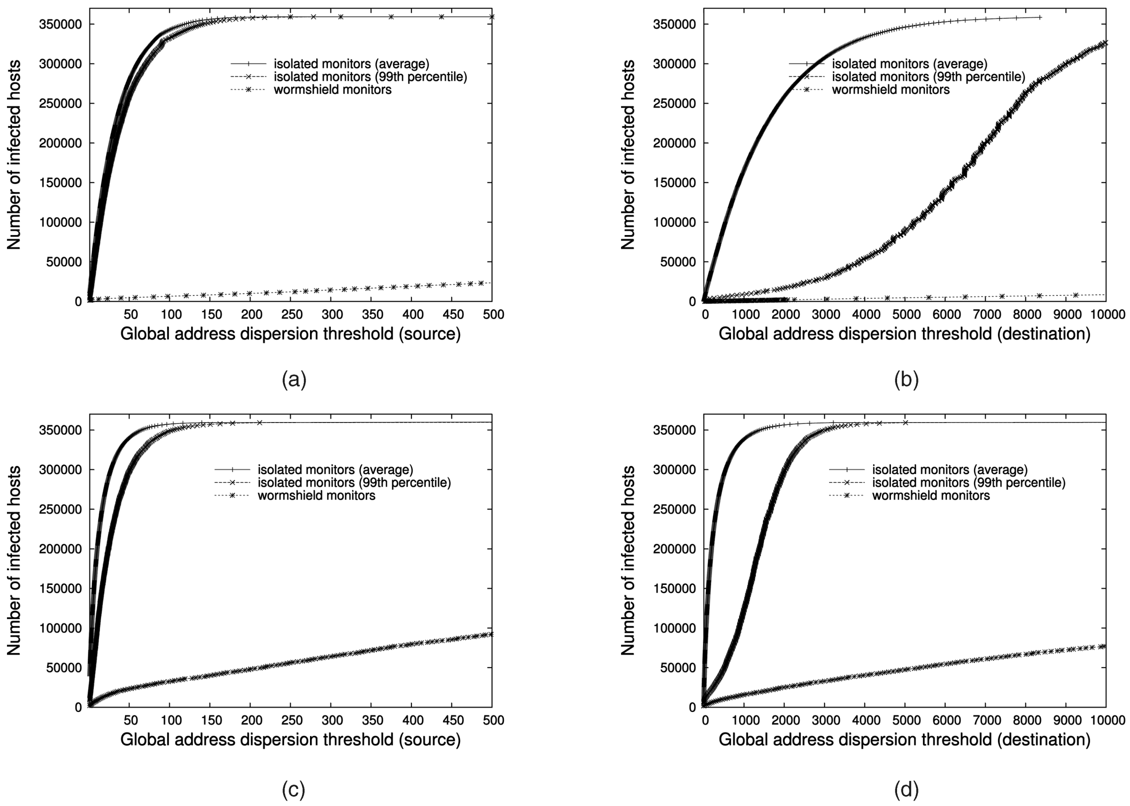


Fig. 8. Comparison of WormShield and isolated monitors in the speed of signature generation for CodeRed worms with varying global thresholds. (a) Source address dispersion (CodRedI-v2). (b) Destination address dispersion (CodeRedI-v2). (c) Source address dispersion (CodeRedII). (d) Destination address dispersion (CodeRedII).

Fig. 11 plots the number of infected hosts at the signature generation time as a function of the number of false signatures by varying the global thresholds, which is essentially the *receiver operating characteristic* (ROC) curves of signature generation speed against false alarms. We observe that for both uniform and subnet-preferred scanning worms, WormShield monitors generate the signature faster and more accurately than isolated monitors. Note that the number of infected hosts shown in Fig. 11 is in log-scale. When there is no false signature, the average and the 99th percentile of isolated monitors generate the signature of CodeRedI-v2 before 231,100 and 11,500 hosts are infected, respectively. In contrast, there are only 1,710 hosts infected before WormShield monitors generate the signature, which is about 135 times faster than the average case of using isolated monitors.

As more false signatures are tolerated, all three cases generate the signature, with less hosts being infected. Fig. 11a shows that, when there are 50 false signatures, the number of infected hosts is reduced to 27,300, 3,600, and 280 for the average of isolated monitors, 99th percentile of isolated monitors, and WormShield monitors, respectively. Fig. 11b shows a similar result for the CodeRedII worm with subnet-preferred scanning. It is important to note that the background traffic is varying from one edge network to another. Therefore, we are not to emphasize on the particular number of false signatures, but rather to show that WormShield monitors are capable of achieving better trade-off between signature generation speed and false positives than isolated ones.

7.5 Deployment Scalability

In this section, we investigate the speed gain of signature generation as more monitors are deployed. We use 200 monitors as our baseline configuration and scale the system size from 200 to 3,000 monitors. We define the *improvement factor* $\gamma(n)$ of signature generation by using n monitors over the baseline configuration as $\gamma(n) = \frac{I_b}{I_n}$, where I_b and I_n are the numbers of infected hosts at the signature generation time for the baseline and n-monitor configurations, respectively. The global thresholds are set as 30 and 3,000 for the source and destination addresses,

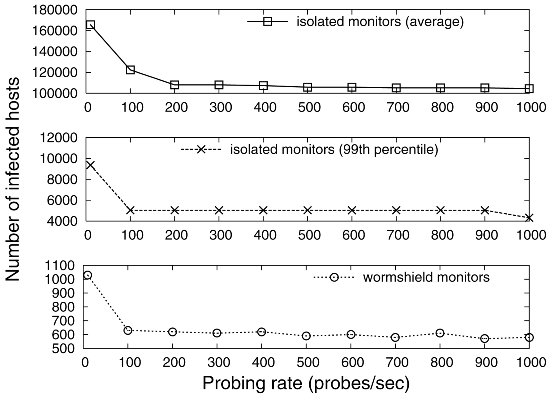


Fig. 9. Number of hosts infected by CodeRedI-v2, with different probing rates under three monitor configurations.

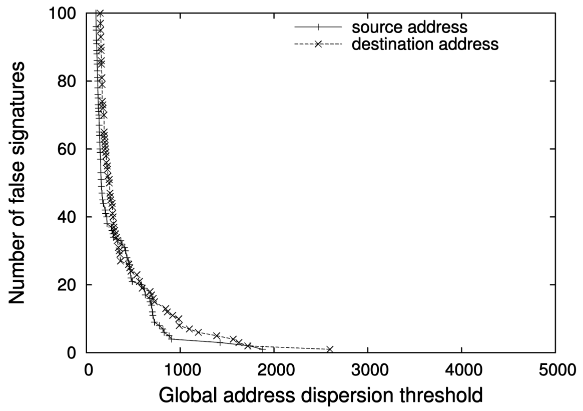


Fig. 10. Number of false signatures drops sharply with increasing global thresholds.

respectively, which eliminates any false signature in the background traffic.

Fig. 12 plots the improvement factor as a function of the number of monitors for the CodeRed worms. Note that we always compare the improvement factors of using an equal number of isolated and WormShield monitors. For the cases of isolated monitors, the signature generation speed does not improve with increasing number of monitors, as demonstrated by the flat lines in Fig. 12. For example, when isolated monitors increase from 200 to 300 for CodeRedI-v2, the number of infected hosts only decreases from 308,800 to 308,100 for the average case and from 42,400 to 22,300 for the 99th percentile case. In contrast, the improvement factor of WormShield increases almost linearly with the deployment scale. The number of infected hosts decreases from 3,610 to 150 when the number of monitored increases from 200 to 3,000, which is a roughly 24 times improvement in signature generation speed, as shown in Fig. 12a. The 3,000 WormShield monitors also gain an improvement factor of 19 over the 256-monitor configuration, as discussed in Section 7.3.

Since the worm fingerprints are not evenly distributed over the edge networks, the improvement factor of WormShield monitors for uniform-scanning worms could be greater than the increase in the number of monitors. This superlinear speedup situation is seen in Fig. 12a. We observe an improvement factor of 24 in signature generation speed when the number of WormShield monitors is 15 times that of the baseline configuration. We also observe in Fig. 12 that the improvement factor of WormShield for uniform-scanning worms is better than that for subnet-preferred ones. This is because uniform-scanning worms generate more inbound and outbound infection attempts that are observable by WormShield monitors.

Thus far, we have compared the performance of WormShield with that of isolated monitors for two variants of CodeRed worms. Our simulation on Slammer worms also shows similar results. The Earlybird system [32] contributes several clever algorithmic designs for the scalable wire-speed implementation of a single worm monitor. However, without sharing the fingerprint statistics among monitors, we anticipate that the Earlybird monitors should have a similar performance as isolated monitors in our simulation.

The Autograph system [16] has a better support for distributed deployment through application-level multicast. Autograph monitors share the IP addresses of port scanners to speed up the classification of suspicious worm flows. However, as in Earlybird, they do not share fingerprint information either, and each Autograph monitor will only accumulate as much worm payload as an isolated one in our simulation. Therefore, with the same global thresholds, we believe that Autograph should have a similar signature generation speed as the best case of isolated monitors. On the other hand, the Autograph monitors use flow-level heuristics to filter out most legitimate traffic and should have better accuracy than isolated monitors or even better than the collaborative WormShield monitors.

8 Extensions and Limitations

Although our simulation results show that collaborative worm signature generation is quite promising, there are several other practical issues that need to be addressed in real-world deployment.

8.1 Privacy-Preserving Signature Generation

It is critical to preserve the privacy of organizations that participate in collaborative worm signature generation.

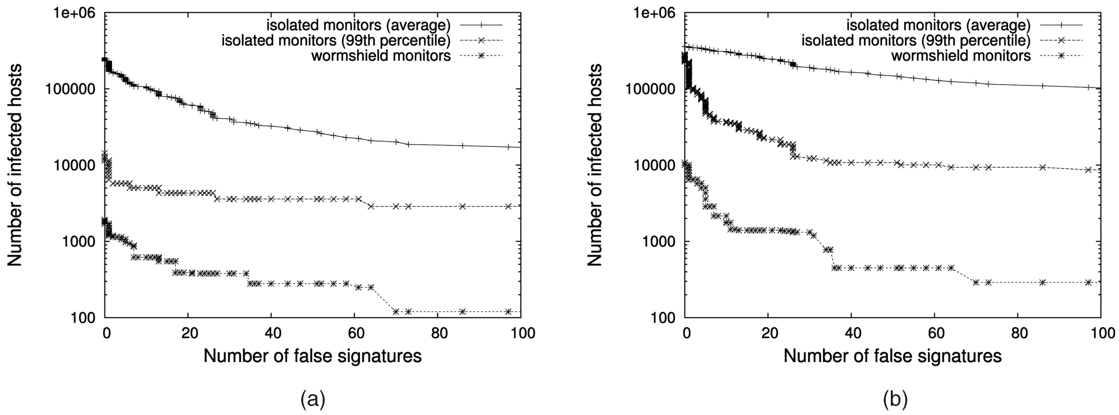
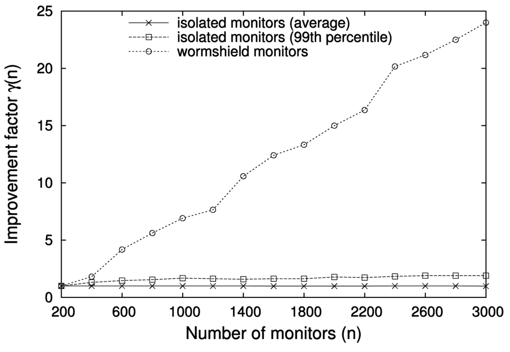
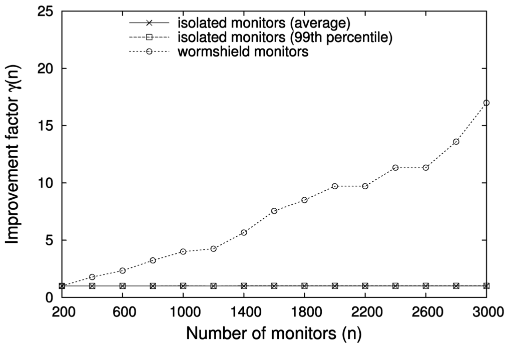


Fig. 11. ROC curves showing the number of infected hosts against the number of false signatures. (a) CodRedI-v2, uniform scanning. (b) CodeRedII, subnet-preferred scanning.



(a)



(b)

Fig. 12. Improvement factors of three signature generation schemes by using an increasing number of monitors from 200 to 3,000. (a) CodeRedI-v2, uniform scanning. (b) CodeRedII, subnet-preferred scanning.

WormShield addresses this issue by anonymizing payload substrings with SHA-1 hash values and by anonymizing IP addresses with cardinality summaries. Each monitor discloses only the SHA-1 hash value of a substring that is above all local thresholds to others, instead of the substring itself. Only the content of a worm signature is revealed to others by the root monitor. In WormShield, each substring is 40 bytes, whereas its SHA-1 hash value is only 20 bytes. Thus, it will be very difficult, if not impossible, for an attacker to recover the substring from a hash value due to the cryptographic property of the SHA-1 hash function. A malicious member may build a reverse mapping from hash values to substrings, which is very expensive (2^{160} entries). However, this mapping does not make the recovery of a 40-byte substring much easier, since each hash value will correspond to 2^{160} possible substrings.

In addition, a 40-byte substring has very limited information that may not be useful for attackers. Since only substrings that exceed local thresholds are subject to global aggregation, it should be more difficult to recover the full packet payload from the disclosed SHA-1 hash values. For IP addresses, only the cardinality summaries are disclosed to other monitors. It is impossible to recover actual IP addresses from a cardinality summary that is indeed an array of counters. Therefore, WormShield monitors generate worm signatures collaboratively without requiring any organization to disclose its private information.

8.2 Security in WormShield Deployment

WormShield relies on a group of distributed monitors working collaboratively to generate worm signatures, and every monitor plays an equally important role in the system. This may attract attackers to target the monitors or even attempt to compromise the entire system. To protect the confidentiality, integrity, and authenticity of the communication among the monitors, information exchange can be encrypted and authenticated by existing means such as using Internet Protocol security (IPsec) or Secure Sockets Layer/Transport Layer Security (SSL/TLS). In addition to leveraging other research works [33], [5], [12] to make Chord more secure and robust, we can introduce a family of consistent hashing functions in WormShield. Each substring will have multiple independent root monitors. Even if the trust relationship cannot be fully established when WormShield is initially deployed, certain measures such as majority voting can be used to resolve possible disputes and fight against collusion among some of the compromised monitors.

The topological information of a WormShield network may also be exploited by peer-to-peer (P2P) worms to propagate very quickly. This is a fundamental threat to any DHT-based system. We anticipate that WormShield monitors are not only built as stand-alone network appliances such as firewalls, but also administrated by security experts. Thus, we hope to minimize the risk of a WormShield monitor being compromised. In our future work, we will investigate the possibility of improving DHT routing schemes to prevent the fast spread of P2P worms. For example, some secret constraints could be predefined to limit the connectivity among nodes. The infected nodes could be detected and isolated, since P2P worms do not know these constraints.

8.3 Polymorphic Worms

Similar to other network-based worm signature generation systems [16], [32], [37], WormShield could be challenged by polymorphic worms [27], [7] with very short or no invariant substrings. Recent studies [27], [7] show that it is very possible for worm authors to implement a polymorphic worm, although it has not yet appeared in the wild [17], [21]. For semipolymorphic worms that have a relatively short invariant substring, window sampling [31] has a higher probability in generating short signatures than using value sampling [32] and Content-based Payload Partitioning (COPP) [16].

The problem of polymorphic worms is orthogonal to the problem of collaborative worm signature generation studied in this paper. Many recently proposed systems for polymorphic worms are either host based [7], [6] or single monitor based [27], [17], [21]. Similar to monomorphic worms, collaborative monitors can speed up the detection of polymorphic worms by observing their global activities. However, instances of a polymorphic worm may not share a common invariant substring of sufficient length for ordinary WormShield monitors. Instead of fingerprinting a worm with its content substrings, Kruegel et al. [17] proposed to fingerprint a worm by the *control flowgraph* (CFG) of the worm executable. The CFG fingerprint characterizes the structural similarities between variations of a polymorphic worm. Therefore, the collaboration techniques in WormShield can be extended to detect polymorphic worms by aggregating the global repetition and address dispersion of CFG fingerprints from distributed monitors.

8.4 Other Practical Limitations

Network address translator (NAT) boxes are often deployed in networks that have limited IPv4 addresses (for example, Europe and Asia). Several internal hosts behind a NAT box share only one global IPv4 address. From a global point of view, all infected internal hosts can be modeled as a single infected host with a higher probing rate. Therefore, the deployment of NAT boxes will not affect the global fingerprint repetition and destination address dispersion, but will reduce the global source address dispersion. In addition, our study in this paper is based on the IPv4 address scheme. The propagation of scanning worms under IPv6 will be extremely slow, since the address space is too sparse, assuming that the total number of hosts on the Internet does not increase suddenly [2]. Instead, IPv6 worms have to leverage other information sources for target discovery. It is relatively straightforward to extend WormShield to IPv6 worms as long as they have exceptionally high fingerprint repetition and address dispersion.

The efficiency of fingerprint filtering in WormShield relies on the Zipf-like fingerprint distribution. Our discussion in Section 4.1 demonstrates that this distribution is inherent to the Internet traffic. However, the filtering scheme will become less efficient if the distribution becomes less skewed. In our future work, we will evaluate the efficiency of fingerprint filtering by deploying WormShield in a real network environment. Moreover, the Zipf-like distribution is a property of the background traffic instead of the worm-attack traffic. Without injecting a significantly large volume of traffic, it would be difficult for a worm attacker to flatten the skewed Zipf-like distribution so as to evade WormShield. On the other hand, injecting too much traffic will saturate the access link, thereby reducing the speed of worm propagation.

9 Conclusions and Future Work

We have presented the WormShield system that collaboratively generates worm signatures by filtering and aggregating fingerprints at distributed monitors in multiple edge networks. First, the exponential growth of global fingerprint statistics was modeled for the early stage of a worm outbreak. Second, the Zipf-like fingerprint distributions were observed in real-life Internet traffic. Therefore, distributed fingerprint filtering reduces the global aggregation traffic by several orders of magnitude. Third, a DAT scheme on Chord was proposed to aggregate the global fingerprint information in a scalable, failure-tolerant, and load-balanced fashion. Fourth, a cardinality-based algorithm was designed to enable the effective counting of all IPv4 addresses in a worm outbreak. Finally, large-scale worm simulations were performed to demonstrate the performance of WormShield in speed gains, signature accuracy, and scalability.

As a continued effort, we are currently developing a WormShield prototype in the C language on both FreeBSD and Linux platforms. It would be interesting to extend the current WormShield work to combat polymorphic worms by examining the structural information of executables in a distributed fashion. We are also investigating a weighted-fingerprint extension of WormShield to apply the flow-level heuristics of scanning worms.

Acknowledgments

The authors are grateful to Xinming He and Shyhtsun Felix Wu for their support on our trace analysis and to Peng Liu, Vern Paxson, and Nicholas Weaver for their helpful discussions. This research was supported in part by the US National Science Foundation ITR Grant ACI-0325409.

References

[1] M. Bailey, E. Cooke, F. Jahanian, and D. Watson, "The Blaster Worm Then and Now," *IEEE Security and Privacy Magazine,* vol. 3, no. 4, pp. 26-31, July/Aug. 2005.

[2] S.M. Bellovin, A. Keromytis, and B. Cheswick, "Worm Propagation Strategies in an IPV6 Internet," *;LOGIN: The USENIX Magazine,* vol. 31, no. 1, pp. 70-76, Feb. 2006.

[3] M. Cai, K. Hwang, Y.-K. Kwok, S. Song, and Y. Chen, "Collaborative Internet Worm Containment," *IEEE Security and Privacy Magazine,* vol. 3, no. 3, pp. 25-33, May/June 2005.

[4] M. Cai, J. Pan, Y.-K. Kwok, and K. Hwang, "Fast and Accurate Traffic Matrix Measurement Using Adaptive Cardinality Counting," *Proc. ACM SIGCOMM MineNet '05 Workshop,* Aug. 2005.

[5] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D.S. Wallach, "Secure Routing for Structured Peer-to-Peer Overlay Networks," *Proc. Fifth Symp. Operating Systems Design and Implementation (OSDI '02),* pp. 299-314, 2002.

[6] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham, "Vigilante: End-to-End Containment of Internet Worms," *Proc. 20th ACM Symp. Operating Systems Principles (SOSP),* Oct. 2005.

[7] J.R. Crandall, Z. Su, S.F. Wu, and F.T. Chong, "On Deriving Unknown Vulnerabilities from Zero-Day Polymorphic and Metamorphic Worm Exploits," *Proc. 12th ACM Conf. Computer and Comm. Security,* Nov. 2005.

[8] A. Czirok, H.E. Stanley, and T. Vicsek, "Possible Origin of Power-Law Behavior in n-Tuple Zipf Analysis," *Physical Rev. E,* vol. 53, no. 6, June 1996.

[9] M. Durand and P. Flajolet, "Loglog Counting of Large Cardinalities," *Proc. 11th Ann. European Symp. Algorithms (ESA),* Sept. 2003.

[10] D. Ellis, "Worm Anatomy and Model," *Proc. 2003 ACM Workshop Rapid Malcode (WORM '03),* pp. 42-50, 2003.

[11] C. Estan and G. Varghese, "New Directions in Traffic Measurement and Accounting: Focusing on the Elephants, Ignoring the Mice," *ACM Trans. Computer Systems,* 2003.

[12] A. Fiat, J. Saia, and M. Young, "Making Chord Robust to Byzantine Attacks," *Proc. European Symp. Algorithms (ESA),* 2005.

[13] K. Hwang, Y.-K. Kwok, S. Song, M. Cai, Y. Chen, and Y. Chen, "DHT-Based Security Infrastructure for Trusted Internet and Grid Computing," *Int'l J. Critical Infrastructures,* vol. 2, no. 4, 2005.

[14] J. Jung, V. Paxson, A. Berger, and H. Balakrishnan, "Fast Portscan Detection Using Sequential Hypothesis Testing," *Proc. IEEE Symp. Security and Privacy,* May 2004.

[15] J. Kannan, L. Subramanian, I. Stoica, and R.H. Katz, "Analyzing Cooperative Containment of Fast Scanning Worms," *Proc. Usenix SRUTI 2005 Workshop,* July 2005.

[16] H.-A. Kim and B. Karp, "Autograph: Toward Automated, Distributed Worm Signature Detection," *Usenix Security,* 2004.

[17] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna, "Polymorphic Worm Detection Using Structural Information of Executables," *Proc. Eighth Symp. Recent Advances in Intrusion Detection (RAID),* Sept. 2005.

[18] A. Kumar, M. Sung, J. Xu, and J. Wang, "Data Streaming Algorithms for Efficient and Accurate Estimation of Flow Size Distribution," *ACM SIGMETRICS Performance Evaluation Rev.,* pp. 177-188, 2004.

[19] J. Li and D.-Y. Lim, "A Robust Aggregation Tree on Distributed Hash Tables," *Proc. MIT Student Oxygen Workshop (SOW '04),* Sept. 2004.

[20] W. Li, “Random Texts Exhibit Zipf’s-Law-Like Word Frequency Distribution,” *IEEE Trans. Information Theory,* vol. 38, no. 6, Nov. 1992.

[21] Z. Li, M. Sanghi, B. Chavez, Y. Chen, and M.-Y. Kao, “Hamsa: Fast Signature Generation for Zero-Day Polymorphic Worms with Provable Attack Resilience,” *Proc. IEEE Symp. Security and Privacy,* May 2006.

[22] M. Liljenstam, D.M. Nicol, V.H. Berk, and R.S. Gray, “Simulating Realistic Network Worm Traffic for Worm Warning System Design and Testing,” *Proc. 2003 ACM Workshop Rapid Malcode (WORM ’03),* pp. 24-33, 2003.

[23] M.E. Locasto, J. Parekh, A.D. Keromytis, and S. Stolfo, “Towards Collaborative Security and P2P Intrusion Detection,” *Proc. Sixth Ann. IEEE SMC Information Assurance Workshop (IAW),* pp. 333-339, June 2005.

[24] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, “Inside the Slammer Worm,” *IEEE Security and Privacy,* Aug. 2003.

[25] D. Moore, C. Shannon, and K. Claffy, “Code-Red: A Case Study on the Spread and Victims of an Internet Worm,” *Proc. Second ACM SIGCOMM Workshop Internet Measurement,* pp. 273-284, 2002.

[26] D. Moore, C. Shannon, G. M. Voelker, and S. Savage, “Internet Quarantine: Requirements for Containing Self-Propagating Code,” *Proc. 22nd Conf. Computer Comm.,* 2003.

[27] J. Newsome, B. Karp, and D. Song, “Polygraph: Automatic Signature Generation for Polymorphic Worms,” *Proc. IEEE Security and Privacy Symp.,* May 2005.

[28] V. Paxson, “Bro: A System for Detecting Network Intruders in Real Time,” *Computer Networks,* vol. 31, no. 23-24, pp. 2435-2463, 1999.

[29] M.O. Rabin, “Fingerprinting by Random Polynomials,” technical report, Center for Research in Computing Technology, Harvard Univ., 1981.

[30] M. Roesch, “Snort—Lightweight Intrusion Detection for Networks,” *Proc. Usenix 13th Systems Administration Conf. (LISA ’99),* pp. 229-238, 1999.

[31] S. Schleimer, D.S. Wilkerson, and A. Aiken, “Winnowing: Local Algorithms for Document Fingerprinting,” *Proc. 2003 ACM SIGMOD Int’l Conf. Management of Data (SIGMOD ’03),* pp. 76-85, 2003.

[32] S. Singh, C. Estan, G. Varghese, and S. Savage, “Automated Worm Fingerprinting,” *Proc. ACM/Usenix Symp. Operating System Design and Implementation,* Dec. 2004.

[33] E. Sit and R. Morris, “Security Considerations for Peer-to-Peer Distributed Hash Tables,” *Proc. First Int’l Workshop Peer-to-Peer Systems,* pp. 261-269, 2002.

[34] L. Spitzner, *Honeypots: Tracking Hackers.* Addison-Wesley, 2002.

[35] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications,” *Proc. 2001 ACM Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm. (SIGCOMM),* 2001.

[36] H.J. Wang, C. Guo, D.R. Simon, and A. Zugenmaier, “Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits,” *Proc. 2004 ACM Conf. Applications, Technologies, Architectures, and Protocols for Computer Comm. (SIGCOMM),* pp. 193-204, 2004.

[37] K. Wang and S.S.G. Cretu, “Anomalous Payload-Based Worm Detection and Signature Generation,” *Proc. Eighth Int’l Symp. Recent Advances in Intrusion Detection,* Sept. 2005.

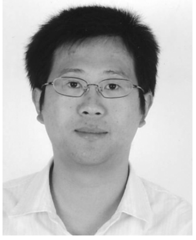
[38] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham, “A Taxonomy of Computer Worms,” *Proc. 2003 ACM Workshop Rapid Malcode (WORM ’03),* pp. 11-18, 2003.

[39] N. Weaver, S. Staniford, and V. Paxson, “Very Fast Containment of Scanning Worms,” *Proc. Usenix Security Symp.,* pp. 29-44, 2004.

[40] K.-Y. Whang, B.T. Vander-Zanden, and H.M. Taylor, “A Linear-Time Probabilistic Counting Algorithm for Database Applications,” *ACM Trans. Database Systems,* vol. 15, no. 2, pp. 208-229, 1990.

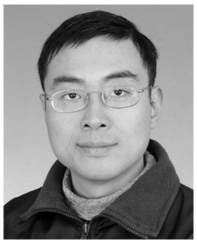
[41] V. Yegneswaran, P. Barford, and S. Jha, “Global Intrusion Detection in the DOMINO Overlay System,” *Proc. Network and Distributed System Security Symp. (NDSS),* 2004.

[42] C.C. Zou, L. Gao, W. Gong, and D.F. Towsley, “Monitoring and Early Warning for Internet Worms,” *Proc. ACM Conf. Computer and Comm. Security,* pp. 190-199, 2003.



Min Cai received the PhD degree in computer science from the University of Southern California in 2006. He joined IBM Research, Beijing, in 2001, where he worked on multimedia networking. His research interests include intrusion detection, peer-to-peer and grid computing, and semantic Web and Web services technologies. He is a member of the IEEE.

Kai Hwang received the PhD degree from the University of California, Berkeley. He is a professor of electrical engineering and computer science and the director of the Internet and Grid Computing Laboratory, University of Southern California (USC). Presently, he leads the US National Science Foundation (NSF)-supported ITR GridSec project at USC. The GridSec group develops security-binding techniques and defense infrastructures for trusted peer-to-peer (P2P) and grid computing. He specializes in computer architecture, parallel processing, Internet security, and grid, P2P, cluster, and distributed computing systems. He is an IEEE fellow.



Jianping Pan received the BS and PhD degrees in computer science from Southeast University in 1994 and 1998, respectively. He is an assistant professor of computer science at the University of Victoria, Canada. He has served as a postdoctoral fellow and a research associate at the University of Waterloo, Canada. His current research interests include protocols for advanced networking, performance analysis of network systems, and applied network security. He is a member of the ACM and the IEEE.

Christos Papadopoulos received the PhD degree in computer science from Washington University, St. Louis, in 1999. He is currently an associate professor at Colorado State University. His interests include network security, router services, multimedia protocols, and reliable multicast. In 2002, he received a US National Science Foundation (NSF) Faculty Early Career Development (CAREER) Award to explore router services as a component of the next-generation Internet architecture. He is a senior member of the IEEE.