# Defending a P2P Digital Preservation System

Bryan Parno and Mema Roussopoulos, *Member*, *IEEE*

**Abstract**—The LOCKSS (Lots Of Copies Keep Stuff Safe) system allows users to store and preserve electronic content through a system of inexpensive computers arranged in an ad hoc peer-to-peer network. These peers cooperate to detect and repair damage by voting in "opinion polls." We develop a more accurate view of how the network will perform over time by simulating the system's behavior using dynamic models in which peers can be subverted and repaired. These models take into account a variety of parameters, including the rate of peer subversion, the rate of repair, the extent of subversion, and the responsiveness of each peer's system administrator. These models reveal certain systemic vulnerabilities not apparent in our static simulations: A typical adversary that begins with a small foothold within the system (e.g., 20 percent of the population) will completely dominate the voting process within 10 years, even if he only exploits one vulnerability each year. In light of these results, we propose and evaluate countermeasures. One technique, Ripple Healing, performs remarkably well. For models in which all system administrators are equally likely to repair their systems, it eliminates nearly systemic levels of corruption within days. For models in which some administrators are more likely to repair their systems, Ripple Healing limits corruption, but proves less effective, since these models already demonstrate superior performance.

**Index Terms**—Distributed applications, protection mechanisms, backup/recovery, model development, libraries/information repositories/publishing, peer-to-peer digital preservation.

## 1 Introduction

In this section, we discuss the general class of peer-to-peer networks and then give a brief overview of the motivation and design goals of the LOCKSS system (we discuss LOCKSS in greater detail in Section 2.1). Finally, we summarize the motivation and contributions of this paper.

### 1.1 Peer-to-Peer Networks

The relatively recent emergence of peer-to-peer networks has introduced a new realm of systems research. Instead of relying on the traditional client-server model of connectivity, peer-to-peer systems make all participants in the system equal and attempt to harness the latent computing power of computers (particularly PCs) distributed across the Internet. From a security standpoint, peer-to-peer systems offer advantages and disadvantages. By removing the concept of a central server, they eliminate any dependency on a single point of failure, making it much harder to bring down the entire system by targeting just one computer (much to the consternation of the RIAA in its battles with music piracy [5]). However, the inherent anonymity[1] and the distributed trust paradigm of peer-to-peer systems (which typically requires peers to trust a large majority of the other peers, rather than one central organization) leave such networks vulnerable to subversion from within. This class of problems encompasses everything from active subversion of otherwise legitimate peers to the well-known Sybil attack, in which a single machine spawns hundreds or even thousands of identities within the system. In fact, Douceur has shown that any system without a logically centralized authority figure will remain vulnerable to Sybil attacks [9]. Thus, as the development of peer-to-peer systems continues, it is important to consider the unique threats posed to such systems.

1. In most peer-to-peer systems, participants use weak identities (such as IP addresses) to communicate. Without face-to-face exchange of strong identities (such as public keys) or other offline mechanisms, peers cannot know exactly who they are talking to in a peer-to-peer system.

- *B. Parno is with the Department of Electrical and Computer Engineering, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213. E-mail: parno@cmu.edu.*
- *M. Roussopoulos is with Harvard University, 33 Oxford Street, Rm 227 Cambridge, MA 02138. E-mail: mema@eecs.harvard.edu.*

*Manuscript received 30 June 2004; revised 15 Dec. 2004; accepted 17 Dec. 2004.*
*For information on obtaining reprints of this article, please send e-mail to: tdsc@computer.org, and reference IEEECS Log Number TDSC-0100-0604.*

### 1.2 LOCKSS: A Digital Preservation System

The LOCKSS system [19] attempts to harness the benefits of a peer-to-peer network while preventing, detecting, or at least slowing attacks based on the system's decentralized nature. LOCKSS primarily helps libraries cope with the ongoing digitization of scholarly materials. Traditionally, libraries have preserved magazines, newspapers, and journals by purchasing subscriptions and then storing physical copies of each issue. With the growth of the Internet, more and more periodicals have moved online, sometimes even to the exclusion of publishing physical copies. As publications shift to an electronic medium, however, libraries often only receive *access* to material, rather than possession of the actual bits. This makes them highly dependent on the publisher. If the publisher discontinues the archival service, raises its rates, or declares bankruptcy, the libraries and their patrons lose access to the periodicals. As an even more insidious threat, the publisher may decide to revise or delete entire portions of a document at some later date. While this concern may sound like Orwellian paranoia, this phenomenon has already occurred in the real world.

In its March 2nd 1998 issue, *Time* magazine published an essay by George Bush Sr. and Brent Scowcroft which explained the United States' decision not to remove Saddam

1545-5971/04/$20.00 © 2004 IEEE Published by the IEEE Computer Society

Hussein from power in Iraq during the first Gulf War. They cite reasons often employed by critics of the invasion initiated by George W. Bush. While the article originally appeared on *Time*'s Web site along with the rest of the issue, it has since disappeared. In fact, the article has been expunged from the online table of contents as well, leaving no hint of its existence. Fortunately, the Memory Hole noted the omission [30] but it seems reasonable to hope for a better system of historical preservation than reliance on observant Web surfers.

LOCKSS' operation closely mirrors Thomas Jefferson's proposal:"...let us save what remains: not by vaults and locks which fence them from the public eye and use in consigning them to the waste of time, but by such a multiplication of copies, as shall place them beyond the reach of accident" [15]. LOCKSS attempts to achieve long-term information preservation by constructing a peer-to-peer system that connects libraries with one another. Given the perennial budget shortfalls at libraries [4] using large RAID arrays and redundant power supplies at each of the libraries is not an option, so LOCKSS is designed with the constraint that the system will run on cheap PC's. Instead of relying on expensive hardware, each LOCKSS peer maintains a digital copy of the electronic resource in question and cooperates in "opinion polls" to detect and repair damage done to the copy. By limiting the rate at which polls are conducted, the system generates inertia that resists an adversary's rapid attempt to infiltrate the system.

LOCKSS also avoids any dependence on long-term secrets, thus excluding the use of digital signatures. In addition to the difficulty that inexperienced administrators often encounter when using digital signatures, we expect LOCKSS to operate on sufficiently long time scales that a determined adversary could extract the secret in question by corrupting an insider, brute-forcing the algorithm, or otherwise subverting an individual system. Instead, LOCKSS relies on its polling mechanism to select random samples from the entire population of the system, so that the adversary can only corrupt the archives by subverting an overwhelming number of peers. LOCKSS also avoids reputation systems since we cannot allow an adversary to accumulate a positive reputation, only to cash in on his efforts with an attack. Since the content preserved by LOCKSS may be considered offensive to some individuals, organizations, or even governments, the system's design must anticipate attacks from powerful adversaries over long periods of time. Rosenthal et al. discuss other characteristics of the expected adversaries, as well as the rationale behind the various design decisions made in the original LOCKSS proposal [27].

### 1.3 Motivation and Contributions

Previous work on LOCKSS focused on a static subversion model. At the start of the lifetime of the system, the adversary subverts some proportion of the peers participating in the system, and these peers remain under the control of the adversary for the lifetime of the system. Given the nature of the Internet, it is unlikely that the subverted peers will remain subverted forever and even more unlikely that unsubverted peers will stay unsubverted for their entire lifetime. In this paper, we develop dynamic models of subversion and repair where, throughout the lifetime of the system, we expect that peers may be repeatedly subverted by the adversary and subsequently healed by their administrators. We consider the possibility of multiple bugs and/or vulnerabilities in the system, affecting different peers at different times; we also analyze human factors, such as the responsiveness of each system administrator, that influence the rate at which peers are repaired. We believe these dynamic models more accurately reflect the unpredictable nature of a distributed system spread across the Internet.

After exploring multiple models, we find that simulations incorporating a dynamic population generally demonstrate significantly worse performance than the original static simulations. For example, in the static simulations, an adversary that begins by subverting 30 percent of the population can only achieve an average corruption level of 65 percent after 40 years, while an adversary that starts with only 20 percent of the population but can exploit one vulnerability each year affecting 30 percent of the population can achieve a 90 percent corruption level within 11 years even in the face of active recovery efforts. Fortunately, in models where system administrators are uniform in their ability to detect and patch vulnerabilities, the use of Ripple Healing can eliminate virtually all traces of corruption within days, even when the adversary begins by subverting 65 percent of the population. In models where different system administrators exhibit different levels of responsiveness, the Ripple Healing technique is less effective, although the system achieves better overall performance. In essence, the Ripple Healing technique uses a viral healing method to counter the adversary's intrusion into the system. If a system administrator suspects a machine has been compromised, he/she alerts all of his/her friends, giving them the opportunity to cleanse and/or update their systems as well. Given the important preservation role played by LOCKSS, libraries have significant motivation to respond to such alerts, and since the alerts arrive via a human network, they have an added level of confidence. However, we recognize that some system administrators cannot or will not respond to such alerts and, so, we incorporate this element into our simulations as well.

The remainder of this paper is as follows: In Section 2, we give a brief overview of the LOCKSS protocol. In Section 3, we describe related work. In Section 4, we measure the effectiveness of an adversary that attempts to modify content without being detected under various models of dynamic subversion and repair in the system. Finally, in Sections 5 and 6, we summarize our results and propose directions for future research.

## 2 Background

### 2.1 LOCKSS Overview

In the LOCKSS system, peers divide their digital collections into archival units (AUs), typically consisting of one year's run of a journal. For simplicity, we will consider the system's operation with only one AU; though, in actual practice, it would maintain hundreds or even thousands of

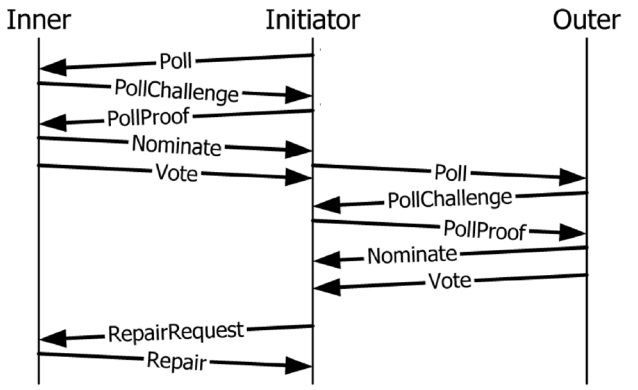


Fig. 1. **Voting Protocol**. This figure shows the messages exchanged between LOCKSS peers participating in an opinion poll. The left side represents an inner-circle peer, and the right side represents an outer-circle peer. Time flows from top to bottom.

separate AUs. Each library in the LOCKSS system is assumed to begin with a list of "friends" who also use the system. These friends represent entities with which the library maintains out-of-band relations. For instance, Harvard's library might include MIT and Stanford on its list. In general, these relationships might form clusters within the system, or they might represent a reasonably random sampling of the population, so that the probability that MIT has Stanford on its friends list is independent of the probability that MIT has Harvard on its list. To simplify our analysis, the simulations presented will assume an unclustered approach, except where otherwise noted.

Each peer also maintains a "reference list" containing other peers it has recently discovered. When a peer joins the LOCKSS system, it initializes its reference list with the contents of its friends list. Periodically,[2] at a rate faster than the rate of natural bit degradation, the peer begins an "opinion poll" by choosing a random sample of peers from its reference list. We refer to this sample as the inner-circle peers. The peer sends a Poll message to each inner-circle peer, inviting it to participate in the opinion poll (see Fig. 1). When an inner-circle peer receives a Poll message, it responds with a PollChallenge message, asking the initiator for a proof of effort based on a fresh, random challenge value. The proof of effort requires the use of memory-bound functions [1] to respond to the challenge, so the protocol forces the poll initiator to exert a considerable amount of computational effort, limiting spurious poll initiation.

After the initiator has successfully responded to each of the inner-circle challenges with a PollProof message, each of the inner-circle peers returns a vote in the form of a secure hash of the AU. Each inner-circle peer also nominates a set of peers from its own reference list. The initiator uses a randomly selected subset of these nominations to form the poll's outer circle. These outer-circle peers are also invited into the poll, with no indication of their status in the outer circle. Outer-circle votes are not used to determine the outcome of the poll. We use the outer circle for discovery purposes, i.e., to expand the list of known peers in the system.

2. Currently, every three months.

Once the initiator has received all of the votes (and assuming it has received enough to reach a quorum), it compares each vote with a hash of its own AU. If an overwhelming number (set as a system parameter[3]) of votes agree with its copy, it assumes the copy remains undamaged, sets a refresh timer of three months on the AU, and goes about its normal activities. If an overwhelming number of votes disagree, then it assumes its copy has been damaged and requests a fresh copy from one of the disagreeing inner-circle peers. That peer will provide a copy only if the poll initiator has successfully participated in an earlier poll called on that AU by the disagreeing peer. This prevents theft since it requires a peer to demonstrate that it possesses a legitimate copy of the AU before it can receive a replacement.

If the vote provides an indeterminate result, the poll initiator suspects either a malfunction or a malicious attack, since it is unlikely that so many peers are simultaneously experiencing natural bit degradation. Instead of attempting to diagnose the problem, the initiating peer raises an Inconclusive Poll alarm, alerting a human operator to the discrepancy. This is an expensive operation, so LOCKSS must carefully balance the importance of security with the danger that false alarms will discredit the system or make it impractical.

After a poll, the initiator updates its reference list to avoid relying on any one set of peers. First, it removes peers that voted in the poll, so that the next poll will use a different sample of the reference list. Second, it replenishes its reference list by inserting all of the outer-circle peers whose votes agreed with the final outcome of the poll. An agreeing vote serves as "payment" to gain entrance into the reference list of the poll initiator; this gives the entering peer the opportunity to potentially influence future polls. Finally, it inserts a small, randomly chosen subset of peers from its friends list. We call this operation "churning." The first two steps limit the long-term accumulation of reputation. This prevents an adversary from agreeing in a few polls in order to gain entrance to the reference list, only to cash in by attacking. The final step, churning, highlights the tension between using only friends when conducting a poll and using unknown peers discovered (nominated) in a previous poll. Churning helps slow the growth of the adversary's presence in the reference list since, during an attack, the friends list tends to remain less corrupt than the reference list. However, we cannot strictly limit the reference list to the friends list since this would give the adversary a static list of target computers and undermine our goal of taking a random sample from the population.

We have briefly described the LOCKSS protocol here. Maniatis et al. provide a more detailed explanation and evaluation of the defenses (such as churning) mentioned above [18], [19].

### 2.2 Potential Adversaries

The design of LOCKSS inherently requires the ability to defend against extremely powerful, patient adversaries attempting to subvert the system. LOCKSS must preserve

3. Currently, the system uses 70 percent as the threshold for an "overwhelming" vote.

data for decades, and it takes little imagination to envision potential attackers. As Orwell notes, "Who controls the past controls the future" [22]. For example, a tobacco company might want to alter the results of a study linking smoking with lung cancer, or a large, powerful company might wish to eliminate an article establishing a competing researcher's patent claim.

Rosenthal et al. present some of the considerations that went into the development of the current adversary model [27]. In doing so, they surpass a large percentage of peer-to-peer systems that assume well-behaved, trustworthy peers or leave security as an area for future work (see Section 3). In this paper, we concern ourselves primarily with the stealth-modification, or "lurking," adversary who wishes to alter documents preserved by LOCKSS while remaining undetected (i.e., without causing an Inconclusive Poll alarm). To accomplish this, he attempts to infiltrate the system by compromising peers in the system, but he continues to vote correctly in all of the opinion polls. When it comes time to recommend outer-circle peers, every compromised peer exclusively recommends other compromised peers. Thus, over time and in the absence of countermeasures, the adversary's presence in the unsubverted peers' reference lists grows and eventually reaches the point where he will have sufficient presence in the polls to convince unsubverted peers that they have a bad copy of the AU in question. When the unsubverted peer requests a repair, the adversary will happily supply his own altered version.

Clearly, LOCKSS may face other types of adversaries as well. A nuisance adversary might try to cause enough spurious alarms to discredit the system. An attrition adversary might use compromised computers to launch a denial of service attack on the system and prevent peers from successfully completing polls. With enough interference of this sort, the AU will be lost through standard bit rot and hardware failures. Additionally, a thief might try to obtain copies of AUs it does not rightfully own. While these adversaries certainly pose a threat to the system, ultimately it is the stealth modification adversary who truly undermines the essence of LOCKSS and, thus, poses the most insidious threat.

### 2.3 Adversary Capabilities

To ensure the long-term viability of LOCKSS, we must design the system to resist an extremely powerful adversary. While LOCKSS may never experience an attack from an adversary with such power, this design strategy forces us to choose conservative techniques that will resist most forms of attack. Thus, we provide the adversary with unlimited identities (since LOCKSS bases identity on IP address, we assume the adversary can purchase or spoof an unlimited number of addresses), perfect work balancing between any of the peers he controls and instant communication between all of the subverted peers in the network. We also assume that the adversary knows all of the system's parameters and can instantly exploit any vulnerability he discovers. Finally, the original LOCKSS paper assumes that the adversary can take over a fixed percentage of peers initially and retain control over them indefinitely [18], [19]. In Section 4, we will explore the effects of altering this assumption.

## 3 Related Work

The original LOCKSS team at Stanford currently supports the existing deployment of LOCKSS at more than 80 institutions worldwide, with the support of publishers representing more than 2,000 titles [24]. They are also investigating stronger measures to combat the attrition adversary, using effort-balancing and admission-control techniques. At Harvard, Greenstadt et al. are investigating the security model of the system, analyzing the trade-offs involved in adding public-key-based authentication to the LOCKSS protocol [14]. Bungale et al. are analyzing the trade-offs between the *consensus-based* version of LOCKSS studied here and a *conservation-based* version [6].

In the peer-to-peer realm, systems such as PAST [28], OceanStore [16], and Intermemory [13] attempt to provide decentralized digital storage. However, in general, these systems provide storage to individual members of the system, rather than collectively attempting to preserve a single document. Thus, one peer's copy of a document in no way benefits the integrity of another member's copy. They also assume that most of the population follows the protocols properly. Furthermore, none of the systems plan for the long term, at least not on the scale necessary for libraries to preserve information for generations to come.

Several papers provide a general survey of security issues facing peer-to-peer systems. Wallach provides an overview of such concerns [31] with a focus on routing and file sharing. Sit and Morris offer a more in-depth exploration of these issues [29] but they limit their analysis to a mostly qualitative look at security in distributed hash tables. These reviews focus on potential security flaws in various systems, but they do not develop a comprehensive adversary model that combines motivation with capabilities.

Wang et al. use an eigenvalue-based approach to study virus propagation in various network topologies [32]. They develop a simple yet effective theory to predict the epidemic threshold for a given network using the network's adjacency matrix. The epidemic threshold represents the critical state beyond which an infection becomes endemic. They simulate propagation by assuming that at each time step, an infected peer may spread the infection to some of its neighbors as well. In our work, we abstract away the adversary's method for subverting peers and assume that he subverts a certain portion of the peers based on the models described below. We also introduce the notion of peer-dependent infection and repair, whereas Wang's work assumes a universal probability of infection and repair. Additionally, we look at the effect of infection on the overall functionality of the peer-to-peer system.

The Wayback Machine, maintained by the Internet Archive [3] takes periodic snapshots of the Internet, largely as a way of preserving digital "cultural artifacts" and providing access to researchers. However, the system requires hundreds of servers with more than 300 TB of data storage. The vast majority of the content is not indexed,

making it difficult to access. Given the vast amount of time that goes into each crawl, it misses considerable amounts of ephemeral data. Also, since the site's spider only accesses free, publicly available sites and obeys robots.txt files requesting that sites not be indexed, the collection only contains the most public of data, not necessarily the material such as academic magazines and journals that are of greatest interest to scholars and whose access requires monetary payment. Finally, the Wayback Machine is inherently a centralized process, the direct opposite of LOCKSS' decentralized approach.

In a process reminiscent of Byzantine fault-tolerance (BFT) schemes (e.g., [7], [8], and [17]), LOCKSS relies on the prevailing opinion of a set of peers, some of which may be controlled by the adversary. However, the scope of the LOCKSS project prohibits the high communication costs entailed by Byzantine fault tolerance. Instead, LOCKSS relies on its polling mechanism to select random samples from the population, eliminating global communication, and knowledge. LOCKSS also uses inertia to slow attacks and includes mechanisms for intrusion detection. Finally, the BFT approach assumes that no more than one-third of the participants are faulty or malicious. In a peer-to-peer system where peers are spread across the Internet, this one-third condition is impossible to guarantee. In LOCKSS, as the number of misbehaving peers increases, the system performance needs to degrade gracefully rather than suddenly and completely. This gives system operators time to detect and deal with a system intrusion.

Approaching the problem from a hardware perspective, the Rosetta Project [11] is creating a 1,000 language corpus and using a microetching technique to preserve the corpus on a nickel disk with an expected life span of 2,000 years. This technique addresses a niche market and has neither the flexibility nor the decentralization of the LOCKSS system. Alternatively, RAID (Redundant Arrays of Inexpensive Disks) allows system administrators to increase the reliability of commodity hard drives. Unfortunately, adding RAID capabilities increases costs while providing little protection from user error, natural disaster, or malicious attacks.

## 4 Simulating Dynamic Populations

### 4.1 Motivation

Previous work testing the performance of LOCKSS has assumed that the system begins in a state in which the adversary, through a virus or implementation vulnerability, has compromised a certain proportion of the population and that the affected peers remain subverted throughout the simulation [19]. A more accurate model would incorporate a population with a dynamic number of compromised peers. When the adversary discovers a new vulnerability in the system, he can exploit it to subvert an entire segment of the population, dramatically increasing his presence in the reference lists of the remaining peers. Currently, the most common forms of exploitation use worms and viruses as vectors, so we refer to each of these instant takeovers as an "infection," and to the subverted peers as "infected." However, over time, we also expect system administrators to detect problems in their systems, perform clean installations and patch existing vulnerabilities. To continue the biological metaphor, we refer to this process as "healing," and to the unsubverted peers as "healthy" or "good." Incorporating these dynamics provides a more realistic simulation of the system's behavior in the real world.

### 4.2 Experimental Setup

To gather simulation data, we use the Narses discrete-event simulator [12]. Narses can accurately simulate networks with a large number of peers over long time periods. It also models the memory-bound computations that LOCKSS uses for its proofs of effort. To compare against results of previous work, we use the same parameters as those used by previous LOCKSS studies [19]. In the simulations, we use a population of 1,000 peers. Each peer has 30 friends and attempts to keep its reference list at a size of approximately 60 peers. Unless otherwise noted, we use a churn rate of 10 percent, meaning that after each poll, we replace 10 percent of the reference list with peers from the friends list. In each simulation, the adversary begins with some percentage of the peers under his control. We also give the adversary as many additional identities (IP addresses) as he needs to execute his attack. These additional identities masquerade as legitimate peers in the system.

The simulations run for 20 years (7,300 days), and the results represent the average of 10 simulations using different random seeds. Standard deviations are less than 2 percent. We present the average reference list corruption as the percentage of a healthy peer's reference list composed of subverted peers. Maniatis et al. [18] show that with a reference list corruption of 65 percent or more, the adversary is able to cause an inconclusive alarm in the system in less than a year, so we aim to keep the corruption well below this level.

In several of the graphs, we plot an average result from the static simulations as a reference point. The static simulation has a constant subversion level of 30 percent and the reference list corruption achieved by the adversary with this constant level of subversion hovers at a little over 60 percent.

### 4.3 Infection

Surprisingly, little documentation exists on the frequency of viruses, worms, and exploits on the Internet. While OpenBSD boasts of "only one remote hole in the default install in more than 8 years" [21], Microsoft releases critical patches on an almost monthly basis. Since LOCKSS is intended to run on multiple platforms, we can assume that a given vulnerability will not affect the entire system, but it becomes difficult to estimate the frequency with which to expect exploits (see Section 4.4 for further discussion). However, one would typically expect to see the same general behavior shown in Fig. 2. This simulation was run with an initial population of 1,000 peers, with 30 percent initially subverted. Infections occur twice a year and affect

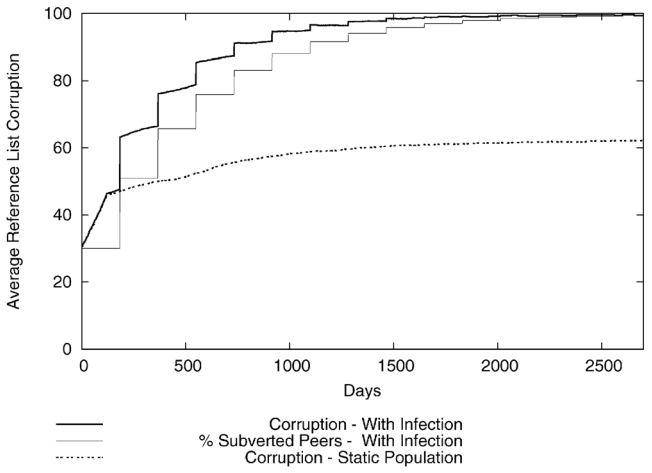


Fig. 2. **The Effect of Infection**. The average reference list corruption (y-axis) as simulation time moves forward (x-axis). In this simulation, the adversary exploits a new vulnerability that affects 30 percent of the unsubverted population every six months (182 days). One sample from the static simulations is shown for comparison. The heavier line indicates reference list corruption and the lighter line indicates the percentage of subverted peers in the system. We will follow this convention in subsequent graphs as well.

30 percent of the unsubverted[4] population. In these simulations, the infected hosts are chosen uniformly at random. In the real world, virus susceptibility may correlate with additional variables, such as the quality and abilities of each peer's administrator or the operating system running on the peer. We attempt to model the former effect in Section 4.7, but we leave more detailed models of the peer itself to future work.

In Fig. 2, the normal reference list corruption level plateaus around 63 percent. However, when we give the adversary the ability to exploit additional bugs and systemic vulnerabilities, we see a dramatic rise in the level of reference list corruption that only plateaus when virtually the entire system has been corrupted. Granted, this simulation uses extremely pessimistic figures, but tweaking these parameters will merely extend the system's lifetime without fundamentally altering the behavior seen here. Notice also that the level of reference list corruption rises faster than the percentage of subverted peers in the system. This results from the adversary's lurking strategy. Since subverted peers always recommend other subverted peers, while unsubverted peers recommend a mixture of subverted and unsubverted peers, the overall corruption of the reference lists should increase even faster than the number of subverted peers in the system. We can also see this effect in the static simulation, in which the reference list corruption rises even when the number of subverted peers remains constant. To calculate how fast the number of subverted peers in the system will grow, we assume that a given vulnerability affects a fixed percentage, $virulence$, of the unsubverted peers. We then derive an expected growth rate for the number of subverted peers $M_t$ in the system after $t$ infections using the recurrence relation

$$M_{t+1} = M_t + virulence * (P - M_t) \quad (1)$$

$$= (1 - virulence) * M_t + virulence * P, \quad (2)$$

where $P$ represents the system's total population. Since a recurrence relation of the general form

$$f(t+1) = af(t) + k \quad (3)$$

has a solution of

$$f(t) = \left(f(0) - \frac{k}{1-a}\right) * a^t + \frac{k}{1-a}, \quad (4)$$

we expect the growth of subverted peers to follow the formula

$$\begin{aligned} M_t &= \left(M_0 - \frac{virulence * P}{1 - (1 - virulence)}\right)(1 - virulence)^t \\ &\quad + \frac{virulence * P}{1 - (1 - virulence)} \\ &= P - (P - M_0) * (1 - virulence)^t \\ &= P - G_0 * (1 - virulence)^t, \end{aligned}$$

where $M_0$ represents the initial number of subverted peers and $G_0$ represents the initial number of unsubverted peers in the system. This indicates that the number of subverted peers rises exponentially, limited only by the size of the system.

### 4.4 Repair

Until recently, few resources existed for tracking either the speed and extent of Internet viruses and worms or the rate at which systems are patched and repaired. Projects such as the Honeynet Project [23] and the Network Telescope [10] maintained by CAIDA (Cooperative Association for Internet Data Analysis) provide some hints of what we can expect to see in the future. The Honeynet Project describes an architecture that masquerades as a normal network vulnerable to attack. Researchers preserve the ability to monitor all network and system activity and, thus, can study the techniques used during an attack. A Network Telescope, meanwhile, consists of a portion of routed IP space that does not expect legitimate traffic. Monitoring this space for unexpected traffic can reveal a network attack in progress or the beginnings of a new worm.

Based on data collected at CAIDA, Moore et al. determined that Internet users responded surprisingly slowly to the Code Red threat [20]. Though a patch was released two weeks before the Code Red worm struck, Moore reports that a third of the vulnerable computers remained unpatched even after the worm spent a month rampaging across the Internet with a considerable amount of accompanying publicity. Fortunately, the rate of repair was front-loaded, so that many systems were patched within the first few days. Before dismissing this dismal performance as a Microsoft issue, we should note Rescorla's analysis [25] of the response to the OpenSSL remote buffer overflow exploit announced in July 2002. Although users

4. As an alternative, we could assume that the exploit affects 30 percent of the *entire* population. This would give the adversary a smaller gain since he would already control some portion of the affected peers. In keeping with the LOCKSS assumption of an extremely powerful adversary, we assume the worst-case scenario in which the adversary can actively target the unsubverted portion of the population.

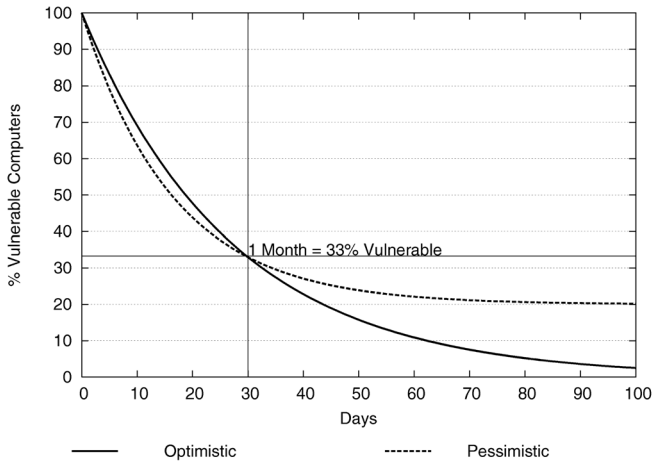


Fig. 3. **Healing Rates**. The percentage of vulnerable peers (y-axis) as time (x-axis) moves forward is shown for the optimistic and pessimistic models of the rate at which computer administrators patch/fix vulnerable peers.

running OpenSSL tend to be more security-conscious and overwhelmingly run some variant of Unix, Rescorla's data show a sluggish response rate very similar to that for the Code Red vulnerability. The use of these results from general "open" systems represents a conservative decision since LOCKSS is a single-purpose system, with a limited number of accessible ports.

Based on the data in these reports, we develop a basic exponential decay function of the form

$$Percent\_Compromised\_Peers = A * e^{-B*days\_elapsed} + C, \quad (5)$$

$$Percent\_Healed\_Peers = 1 - Percent\_Compromised\_Peers \quad (6)$$

to model the rate at which machines are repaired.[5] We consider both pessimistic and optimistic versions of the model. For the pessimistic model, we choose values for $A$, $B$, and $C$ [6] so that 33 percent of the peers remain subverted after a month, with this value tapering off to 20 percent, meaning that the system never fully recovers. For the more optimistic model, we choose constants[7] that still leave 33 percent of peers vulnerable after one month, but eventually bring the level of vulnerability close to 0 (see Fig. 3).

As shown in Fig. 4, the selection of the healing model makes a significant impact on the system's performance. With the pessimistic model, only 80 percent of the infected peers ever recover from a given infection, so every infection gives the adversary a net gain in the percentage of subverted peers in the system, and the quality of the reference lists degrades rapidly. With the optimistic model, the system can recover from most of the damage (approximately 99 percent of infected peers recover) and, thus, valiantly resists the adversary's encroachments. We assume that sufficient time elapses between infections to allow the healing to take effect; otherwise, the optimistic model would merely devolve into the pessimistic model. These data demonstrate the importance of widespread deployment of patches and repairs since rapid deployment to a limited subset of the population provides far less security. Fortunately, in the current LOCKSS network, a carefully designed intrusion response plan enabled team members to patch 95 percent of the deployed systems to fix a (hypothetical) vulnerability within 48 hours [26] indicating a pattern closer to the optimistic model. However, as the system grows and evolves, it will be difficult to maintain this efficiency. Also, both the optimistic and the pessimistic models indicate a similar trend towards increased reference list corruption, due to the growth in the number of subverted peers in the system. Using an optimistic model slows this growth, but the system ultimately displays the same behavior. Taking a conservative stance, the simulations presented will use the pessimistic model unless otherwise noted.

### 4.5 Healing Recurring Infections

To examine the effectiveness of healing in the face of repeated infections, we ran several simulations while varying the percentage of the population affected by each exploit (Fig. 5), the rate at which exploits occur (Fig. 6), and the initial level of systemic subversion (Fig. 7). Since the authors of the original LOCKSS paper [19] note the importance of churning, we also ran simulations to analyze the extent to which churning helped resist the effects of infection (Fig. 8). The primary results concerning the average corruption of the reference lists are presented in Table 1, and we discuss the results in detail below.

Increasing, the percentage of the population affected by each exploit clearly accelerates the overall reference list corruption of the system, particularly after the first few years. The simulations in Fig. 5 were run with an initial subversion level of 30 percent, a 10 percent churn rate, the pessimistic healing model, and an infection once a year. At first, due to the healing of the initial subversion, the system outperforms the static population simulation. However, as the infections recur, the percentage of subverted peers increases, since the healing process never quite eradicates all traces of subversion. Thus, the average corruption of the reference lists catches and then surpasses the static model, with a moderate amount of variation based on the actual percentage of peers affected by each infection. All of the reference list corruption levels eventually slow their growth rate as they approach saturation level.

Varying the rate at which compromises occur also has a serious impact on the system's performance. The simulations in Fig. 6 were run with an initial subversion level of 30 percent, a 10 percent churn rate, the pessimistic healing model, and infections that affected 30 percent of the population. The frequency of infection was varied from twice a year to once every two years, though some results have been omitted for clarity. Once again, the percentage of subverted peers in the system grows in a stepwise fashion. While the frequency of infection clearly has an impact on the system, even the system with infections occurring only

5. We measure $Percent\_Compromised\_Peers$ and $Percent\_Healed\_Peers$ in percentage points, so total compromise corresponds to $Percent\_Compromised\_Peers = 100$.

6. $A = 80$, $B = 0.0606$, and $C = 20$.

7. $A = 99.99$, $B = 0.0369$, and $C = 0.01$.

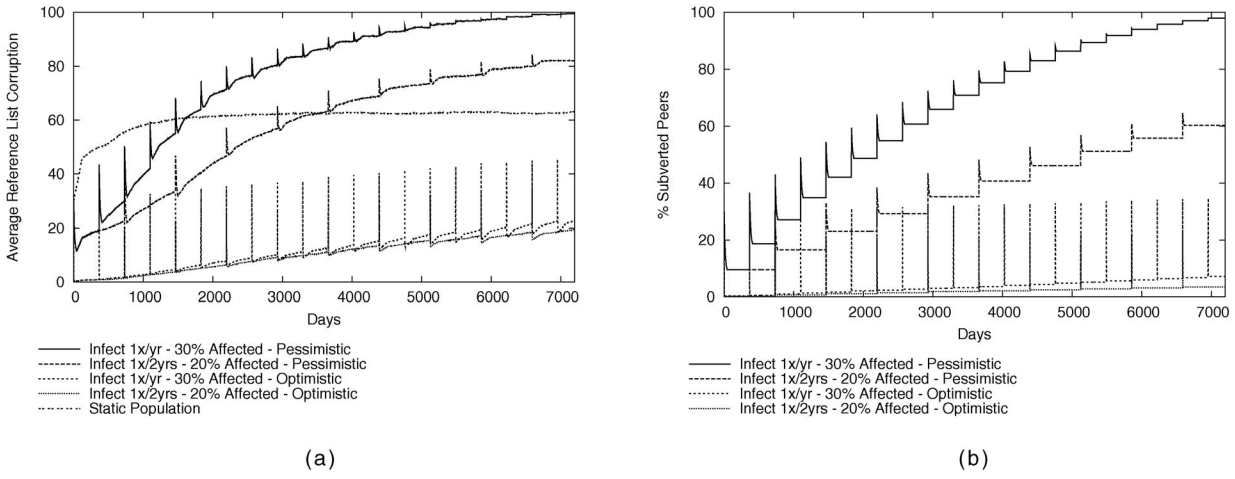


Fig. 4. **Impact of Optimistic versus Pessimistic Healing Models**. (a) Corruption. (b) % Subverted Peers. The average reference list corruption (y-axis) versus simulation time (x-axis). Using a healing model in which most users (approximately 99 percent) eventually patch their vulnerable system significantly improves the system's performance. Representative sample simulations shown.

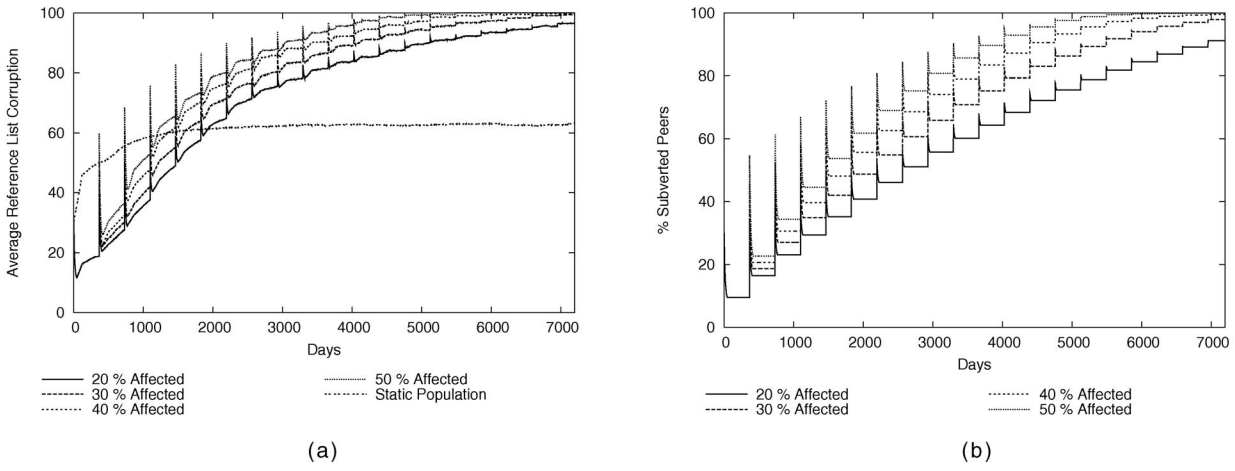


Fig. 5. **Varying Exploit Impact**. (a) Corruption. (b) % Subverted Peers. Varying the percentage of peers affected by each exploit changes the behavior of the average reference list corruption of the good peers' reference lists. One sample from the static simulations is shown for comparison.

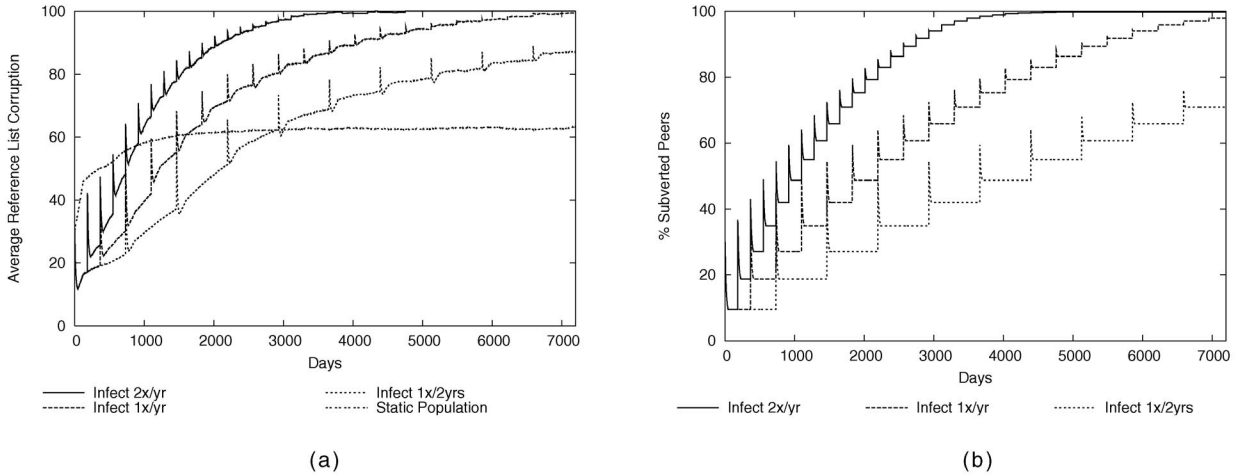


Fig. 6. **Varying Infection Rate**. (a) Corruption. (b) % Subverted Peers. Decreasing the rate of infections decreases the average corruption of the reference lists, but the general trend towards universal reference list corruption remains. One sample from the static simulations is shown for comparison.

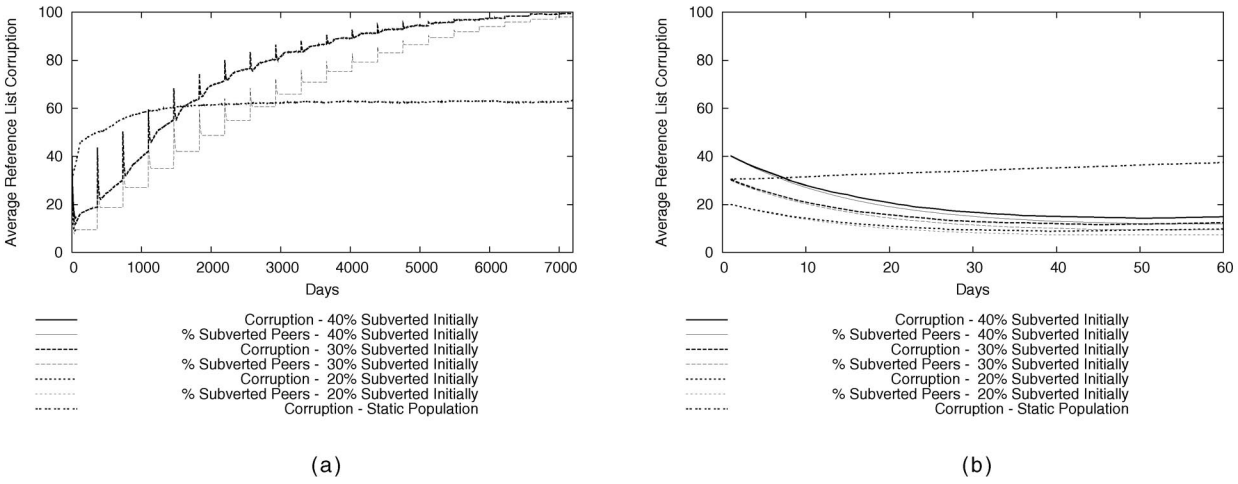


Fig. 7. **Varying Initial Subversion**. (a) Varying initial subversion. Varying the initial subversion of the population has little effect on the overall growth of reference list corruption (to the extent that the various lines merge). Unfortunately, all of the dynamic simulations tend toward systemic reference list corruption. One sample from the static simulations is shown for comparison. (b) Varying initial subversion—zoomed in. On a smaller time scale (the first 60 days of the simulation), we see that despite varying levels of initial subversion, all simulations converge toward the same level of reference list corruption by the 60th day.

once every two years surpasses the static population's reference list corruption level after eight years, and by the end of the simulation (20 years), the corruption has completely overwhelmed the system.

Interestingly, varying the level of initial subversion in the system has little impact on a dynamic population (see Fig. 7a). For these simulations, we held the rate of infection and the percentage of the population affected constant (at once a year and 30 percent affected per exploit, respectively) and varied the initial subversion from 20 percent to 50 percent without much noticeable effect. The reason for this behavior becomes clear when we examine the initial behavior of the system on a smaller time scale, specifically looking at the first 60 days (see Fig. 7b). Since the first new infection does not hit immediately (until the 365th day), the system has plenty of time to heal most of the initially subverted peers, giving each simulation a more or less identical starting point once the infections begin recurring in earnest.

As shown in Fig. 8, increasing the percentage of peers that are churned into the reference list from the friends list reduces the average level of corruption in the reference list. We used an initial subversion of 30 percent, infections that affected 30 percent of the unsubverted population and a pessimistic healing model. Churning has a smaller absolute effect when the rate of infection is increased (from once every two years to once a year in the figure) since the churning effect is drowned out by frequent infestations. Overall, churning certainly helps the system, but it cannot prevent systemic reference list corruption; instead, it merely slows its growth.

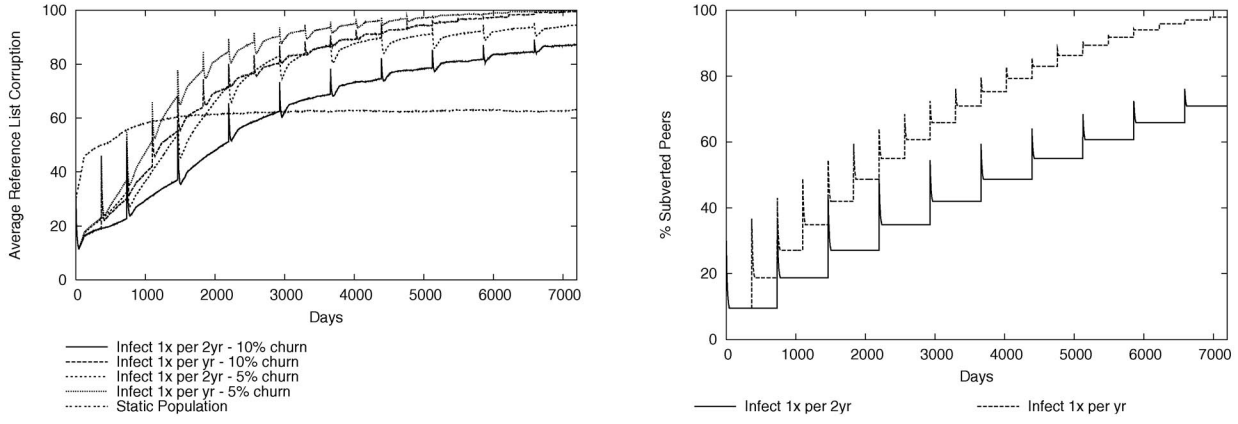


Fig. 8. **Varying Churn Rate**. (a) Corruption. (b) % Subverted Peers. Increasing the percentage of peers churned into the reference list from the friends list reduces the average level of corruption in the reference list. Unfortunately, all configurations tend toward almost total corruption. Increasing the churn rate slows the growth of corruption, but it also makes the friends list a tempting static target for the adversary. One sample from the static simulations is shown for comparison.

TABLE 1
Summary of Simulation Results for Varying Parameters

| Figure | Percent Affected | # Infections Per Year | Initial Level of Subversion | Churn | Reference List Corruption | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | 1 Yr | 5 Yrs | 10 Yrs | 15 Yrs | 20 Yrs |
| 5 | **20** | 1 | 30 | 10 | 19 | 58 | 81 | 91 | 97 |
| 5 | **30** | 1 | 30 | 10 | 19 | 64 | 86 | 96 | 99 |
| 5 | **40** | 1 | 30 | 10 | 19 | 70 | 90 | 99 | 100 |
| 5 | **50** | 1 | 30 | 10 | 19 | 73 | 94 | 99 | 100 |
| 6 | 30 | **2** | 30 | 10 | 26 | 85 | 99 | 100 | 100 |
| 6 | 30 | **1** | 30 | 10 | 19 | 64 | 86 | 96 | 99 |
| 6 | 30 | **0.5** | 30 | 10 | 19 | 45 | 69 | 81 | 87 |
| 7 | 30 | 1 | **20** | 10 | 23 | 65 | 88 | 96 | 99 |
| 7 | 30 | 1 | **30** | 10 | 19 | 64 | 86 | 96 | 99 |
| 7 | 30 | 1 | **40** | 10 | 15 | 64 | 87 | 95 | 99 |
| 8 | 30 | **0.5** | 30 | **10** | 19 | 45 | 69 | 81 | 87 |
| 8 | 30 | **1** | 30 | **10** | 19 | 64 | 86 | 96 | 99 |
| 8 | 30 | **0.5** | 30 | **5** | 23 | 60 | 87 | 91 | 94 |
| 8 | 30 | **1** | 30 | **5** | 23 | 78 | 94 | 98 | 100 |

*This tables illustrates the average reference list corruption over time for various parameter choices. Values in bold indicate the parameter varied in that experiment. For a more detailed graph, see the corresponding figure.*

### 4.6 Fighting Infection

The results above indicate that even when we allow system administrators to patch their systems, the average level of reference list corruption still reaches unacceptable heights. To combat the general trend toward systemic reference list corruption, we examine the effects of two variants of the usual healing protocol.

#### *4.6.1 Clean Start*

In the Clean Start variant, whenever a peer is healed, we alter its reference list to include only unsubverted peers. This simulates a system administrator realizing his system has been compromised, fixing it, and then removing any suspicious peers from his reference list. As illustrated in Fig. 9, Clean Start creates drastic variations in the level of reference list corruption in the system, as the healing efforts battle with the repeated infections. In these simulations, we begin with 30 percent subversion and assume infections occur once a year and affect 30 percent of the unsubverted population each time. All of the good peers use a 10 percent churn rate. Overall, the corruption rises much more slowly than in comparable non-Clean-Start simulations and generally maintains a curve similar to that of the static population, indicating that this technique does help provide an edge to the good peers in the system. The technique has a smaller impact when using the optimistic healing model, largely because the optimistic model keeps the level of corruption so tightly constrained on its own that the Clean Start mechanism can only offer marginal improvements. In practice, it would be difficult for a system administrator to completely purify the reference list. However, by eliminating peers with whom he/she does not have an offline relationship (e.g., by reverting to the peers on the friends list), the administrator will effectively reduce the corruption of the reference list. More importantly, the success of the Clean Start technique provides inspiration for the Ripple Healing technique described below.

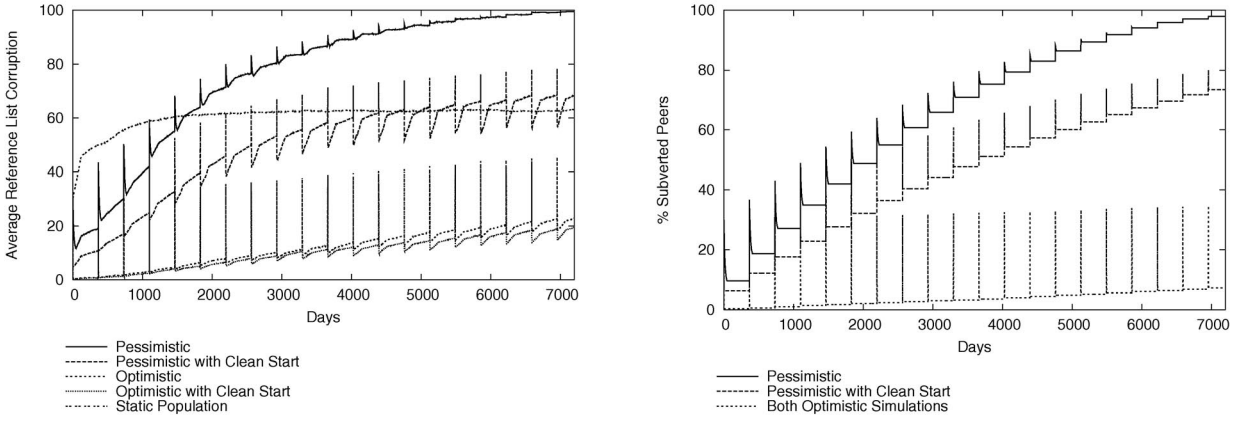


Fig. 9. **Effect of the Clean Start Technique**. (a) Corruption. (b) % Subverted Peers. Giving each healed peer a purified reference list improves performance, despite oscillations. Simulations were run with 30 percent initial subversion and assumed that infections occur once a year and affect 30 percent of the population. All the good peers use a 10 percent churn rate.

#### 4.6.2 Ripple Healing

The other modification we investigate, Ripple Healing, initially seems quite similar. A peer that discovers it has been compromised keeps the same reference list as before, but sends an alert to all of the peers on its friends list. The next day, each of these peers also heals itself if it has been subverted, but it does not continue to propagate the alert to its set of friends. This simulates a system administrator realizing his system has been compromised and alerting his friends that they should investigate their systems too. This could take the form of standard forensic techniques like examining logs or it might cause the friends to apply the latest patches, upgrade their systems to the latest version, or simply reboot the host computer. Regardless of the actual techniques employed, we expect this technique to reduce the number of peers under the adversary's control. Indeed, we omit a graph for this case since, even with 60 percent initial subversion and the pessimistic healing model, the Ripple Healing cured 100 percent of the peers within three days of an infection, leaving the adversary without any foothold within the system. On the one hand, this approach exaggerates the communicativeness and responsiveness of system administrators (we explore more realistic models in Section 4.7). On the other hand, it offers a powerful argument in favor of distributed, automated threat detection systems that would allow computer networks to recognize an incursion and instantly alert the rest of the network to it. However, any system of this sort must guard against manipulation by the adversary, particularly in the form of false alarms. Convincing one or more peers that a vulnerability exists, even if it does not, could set off a flood of warnings and updates throughout the network. The nonautomated version proposed suffers from the relatively slow and unreliable responses of system administrators, but it has the benefit of providing a higher level of authentication that prevents an adversary from manipulating the Ripple Healing system since we expect the alerts to take the form of an email or even a phone call from a known and trusted friend.

To gain further insight into this technique, we develop a mathematical model to describe the number of subverted peers in the system. We define the following variables:

- $M_t$ = number of subverted peers at time $t$.
- $P$ = total population.
- $F$ = size of friends list.
- $H$ = number of peers healed in the normal healing model in one unit of time.

Each of the $H$ peers to be healed has $F$ friends, and assuming an even distribution of subverted peers, $F * \frac{M_t}{P}$ of the friends have been subverted. Ripple Healing will heal them all, so we can create a recurrence relation such that:

$$M_{t+1} = M_t - H * \frac{F * M_t}{P} = \left(1 - \frac{HF}{P}\right) M_t. \quad (7)$$

For this analysis, we focus on the impact of Ripple Healing, so we ignore the effects of the normal healing process (which would subtract an additional $H$ peers from the right-hand side of (7)). Drawing on our work from Section 4.3, we know that a recurrence relation of this form has the solution:

$$M_t = M_0 \left(1 - \frac{HF}{P}\right)^t. \quad (8)$$

Using the standard approximation that $(1 - \frac{1}{m})^n \approx e^{-\frac{n}{m}}$, we can rewrite this as:

$$M_t = M_0 * e^{-\frac{HF}{P}t}. \quad (9)$$

For a fixed population size, this equation indicates that, when the system uses Ripple Healing, the number of subverted peers decays exponentially, with a speed based both on the number of peers healed each day and on the number of friends the average peer possesses. Fig. 10 illustrates the effect of varying these two parameters. All of the variations shown heal virtually the entire system in less than three weeks. Given that in the real system each peer has an average of 30 friends, and that even in the pessimistic model, the system heals an average of 30 peers per day for the first three weeks, we can begin to understand the dramatic impact of Ripple Healing. Intuitively, Ripple Healing mimics the technique used by the worms and viruses to spread the original infection, so the technique can repair the damage almost as quickly as it can be inflicted.

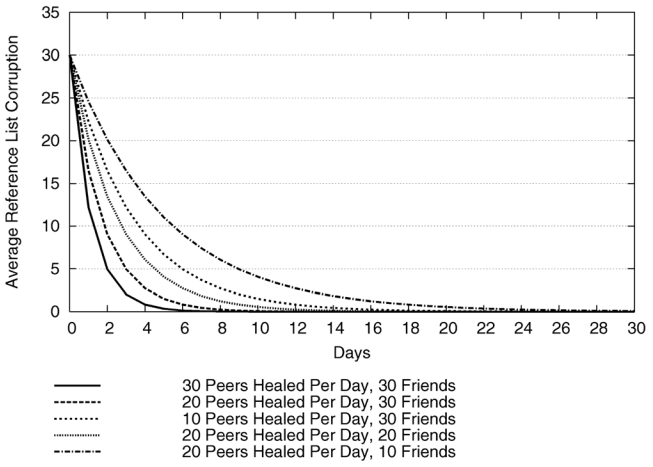


Fig. 10. **Effect of Ripple Healing**. The impact of Ripple Healing depends on the number of friends each peer has, as well as on the number of peers healed each day.

### 4.7 An Alternate System Model

In the preceding simulations, we have assumed that both the probability of becoming infected and the probability of being healed are independent of the peers involved. In other words, at each stage, we decide a certain portion of the peers will be infected/healed and then randomly select those peers from the appropriate portion of the population. However, this system may not provide the best model of real-world behavior. In practice, some system administrators remain constantly vigilant, checking on the latest system patches and monitoring their systems' behavior for suspicious activity. Presumably peers with such active

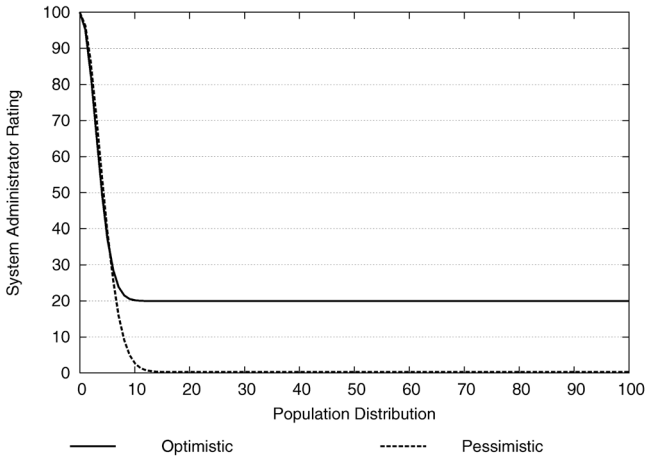


Fig. 11. **System Administrator Abilities**. A few peers have highly skilled system administrators, but the majority have mediocre ratings. The x-axis shows the distribution of skill ratings, indicating for example that in the pessimistic model, approximately 9 percent of the population has a rating of 10 or above.

administrators will prove less susceptible to virus attacks and more likely to detect and repair infections when they occur. Conversely, those administrators lacking the time, interest, or skill to properly administer their systems will have peers that become infected more often and remain infected for longer periods of time. Unfortunately, as Rescorla [25] and Arbaugh [2] note, the vast majority of system administrators tend to fall into the latter category. Systems with known vulnerabilities remain unpatched for months or even years after the initial announcement, even when features like Microsoft's Automatic Update attempt to download and install patches in the background. We might choose to further group peers according to their operating system and assign infection and healing rates to each operating system. However, we choose to focus on the system-administrator model and leave further exploration to future work.

In our new model of the world, we assign each peer a system-administrator rating (with a high rating representing a skilled administrator), following a distribution that gives a few peers a high rating and the vast majority a relatively low rating (see Fig. 11). In other words, we assign a skill level for each peer by randomly selecting a value on the x-axis and then assigning the corresponding rating from the y-axis. To test the impact of these ratings, we repeated the previous experiments with both optimistic and pessimistic models. Initially, we used the system-administrator rating only to determine the probability on any given day that a subverted peer discovers it has been infected, using the formula:

$$P_{discovery} = \frac{sys\_admin\_rating}{100}. \tag{10}$$

This means that in our optimistic model, most peers have a probability of discovering they have been infected within 5 days of infection (since they have a rating of 20 on average, they will have $P_{discovery} = \frac{1}{5}$). Given the preceding discussion, this figure seems closer to an ideal world than the real one. The pessimistic model comes closer to reality as noted in recent studies [27] since most peers have a rating of $0.274 \approx \frac{100}{365}$ and, thus, heal only once a year (since this rating provides $P_{discovery} = \frac{1}{365}$ on any given day).

We then ran further simulations in which the probability of infection during a given attack also depended on the system administrator's rating, such that:

$$P_{infection} = \frac{100 - sys\_admin\_rating}{100}. \tag{11}$$

Fig. 12 shows a sampling of the results. We ran the simulations with infections occurring once per year. The system started with 30 percent of the population subverted and used a 10 percent churn rate. Some simulations used the system-administrator model to determine the probability of both infection and healing, while others only used it to determine the peers to heal, while continuing to use the

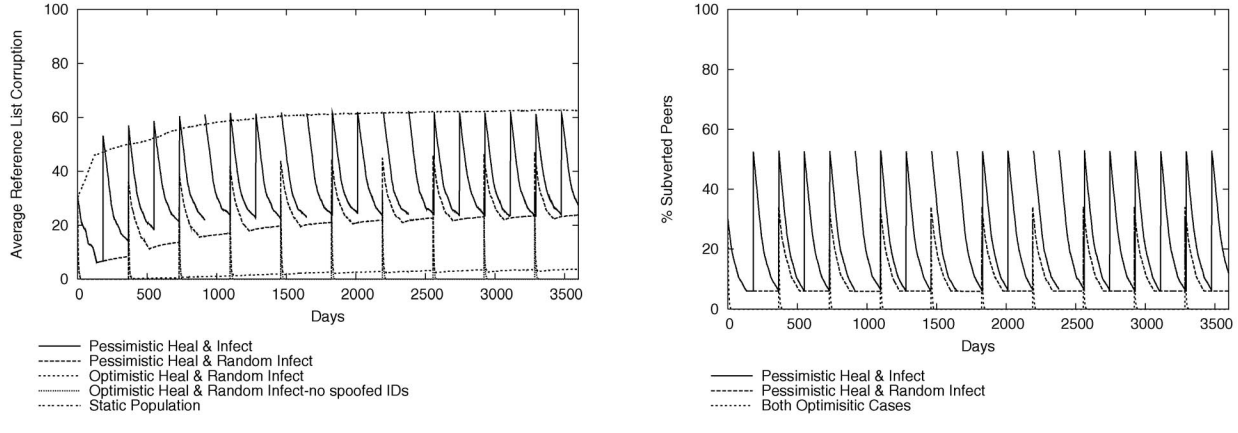


**Fig. 12. Infection and Healing Based on System Administrator Abilities**. (a) Corruption. (b) % Subverted Peers. This graph illustrates the effects of the optimistic and pessimistic system-administrator-based infection and healing models, as well as the performance of models that used system-administrator-based healing with randomized infection. Removing the spoofed IDs improves the corruption level, but both optimistic simulations have the same percentage of subverted peers since we do not count the spoofed IDs as subverted peers.

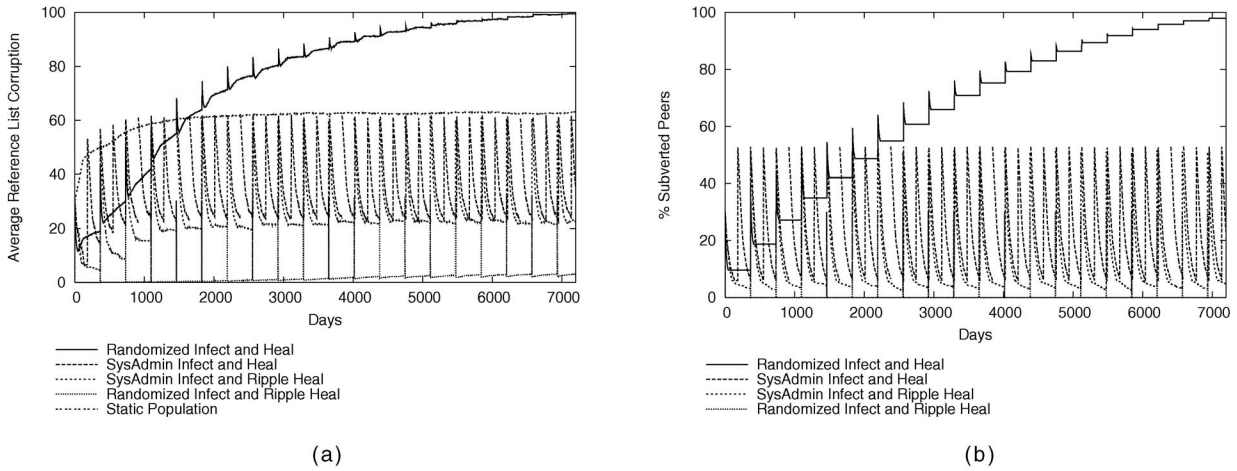


Fig. 13. **Effects of Ripple Healing**. (a) Corruption. (b) % Subverted Peers. Ripple Healing provides only a marginal improvement when used with the system-administrator model, unlike the huge gains it gives the randomized model. In the simulations shown here, we illustrate the difference between using the randomized infection/healing model with and without Ripple Healing, and using the system-administrator model with and without Ripple Healing.

randomized infection model. For runs that did not use the system administrator's rating to determine infections, each infection randomly affected 30 percent of the unsubverted population. Once again, we show the average reference list corruption level of a static population as a baseline. Looking at the optimistic cases, we see that the reference list corruption remains extremely low, which makes sense given that in this case even the worst system administrators heal their peer every five days.

However, while the reference list corruption level with optimistic healing remains remarkably low, it does slowly increase, which seems surprising in light of the fact that the entire population heals itself within 5 days. The explanation lies in the adversary's ability to spoof additional identities. Recall that in addition to starting with control of a portion of the peers in the population (30 percent in this case), we also give the adversary additional identities that can masquerade as legitimate peers. To confirm this explanation, we repeated the optimistic simulation without the extra identities and, in this case, the reference list corruption level did indeed go to zero (except for the spurious spikes near an infection). This emphasizes the importance of designing a system to defend against a Sybil attack [9] since even if the adversary cannot control any other peer in the system, he can still subvert the process by using multiple identities to influence the voting.

In the pessimistic case, it takes the peers much longer to heal after an infection, so the spikes become larger and more pronounced, no longer a transitory phenomenon. Obviously, the pessimistic case performs worse than the optimistic (though still better than the static case), but more interestingly, the simulation that bases infection on the system administrator's abilities shows more reference list corruption than the randomized model of infection. Intuitively, this makes sense since in the system-administrator model, the peers most vulnerable to infection are also the ones least likely to heal themselves, so each successive infection should have a larger impact on the system's level of subversion and hence on the average level of reference list corruption.

Finally, given the extremely successful use of Ripple Healing in the randomized healing model (Section 4.5), we ran experiments using the system administrator's rating to determine the effect of the Ripple Healing. We followed the same procedure outlined above, but when the newly healed peer cycles through its friends list alerting his friends about the infection, the friends respond with a probability based on their system administrator's rating. In other words, good system administrators will immediately respond to such an alert, whereas poor system administrators may simply ignore it. The results differ significantly in comparison with the randomized-infection model (see Fig. 13). Indeed, in the system-administrator model, the Ripple Healing technique provides a marginal improvement at best. In the randomized-infection model, every peer responds instantly, whereas in the system-administrator model, the average peer will typically ignore the warning. Furthermore, the peers most likely to respond to an alert (those with high system administrator ratings) will also be the ones most likely to have already healed themselves and, similarly, the peers least likely to have healed themselves are also those least likely to respond to an alert.

### 4.8 Implications

The results described in the previous two sections illustrate the importance of accounting for human factors in analyzing the behavior of systems in the real world. Assuming a universal pattern of behavior for the entire population of users in a peer-to-peer network may not necessarily create an accurate model of the world. Furthermore, the choice of model creates significant changes in the system's observed behavior. In addition to its more realistic configuration, the system-administrator model generally tends to perform better than the randomized model. Fig. 14 shows the average corruption of the reference lists for varying rates of infection in both models, starting with an initial subversion level of 30 percent and using a 10 percent churn rate. In the simulations using the randomized model, each infection

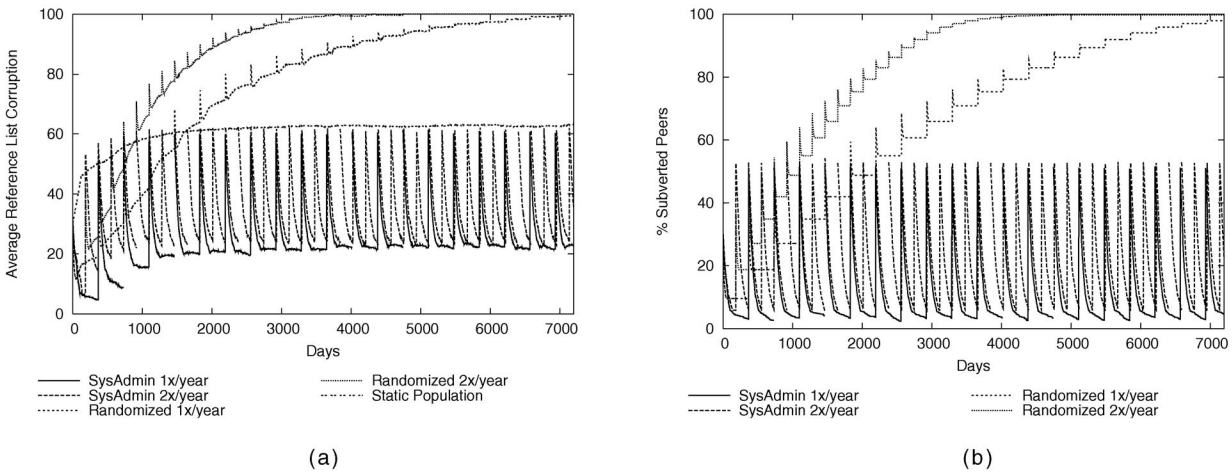


Fig. 14. **Randomized versus System Administrator Systems**. (a) Corruption. (b) % Subverted Peers. A comparison of the randomized and system-administrator models for various rates of infection. In general, the system-administrator model demonstrates better performance (i.e., less average reference list corruption). In the legend, the entries indicate which model of healing/infection the system used (static population, randomized or system-administrator) and the frequency of the infections. For the randomized model, the infections affected 30 percent of the unsubverted peers.

affected 30 percent of the unsubverted peers. This clearly results from the targeted nature of the system-administrator model. The core group of competent system administrators continues to resist or heal the infections indefinitely, whereas in the randomized model, everyone eventually succumbs. On the other hand, the Ripple Healing technique is less effective in the system-administrator model, for the reasons given above. Thus, selecting an appropriate model for the system will help predict its behavior and the techniques that may successfully combat reference list corruption. Furthermore, these results emphasize one of the more unique aspects of peer-to-peer networks: the inherently interdependent nature of the peers within the system. Unlike traditional systems in which a system administrator can concern himself or herself exclusively with his or her own machine's defenses, in a peer-to-peer network each peer necessarily depends on the other members of the network, so a system administrator must worry about the security of all of the other peers in the system. As illustrated above, failure to widely deploy patches for known vulnerabilities can lead to systemic subversion, making the network unreliable even for peers who have successfully patched the vulnerability.

## 5 Future Work

Clearly, additional data on the frequency and severity of viruses, worms and other system compromises in the real world will allow us to improve our models and provide better predictions of LOCKSS' performance. We would also like to explore more detailed models that assign each host an operating system and a particular version of the LOCKSS software and then base each peer's probability of infection on its particular configuration. In addition, the remarkable success of the Ripple Healing mechanism suggests that further investigation into distributed and automated exploit tracking and repair could significantly improve the performance of LOCKSS. The proposed system relies on the human operators to spread the Ripple Healing alerts. This has the advantage of providing a reasonable level of authentication, but the disadvantage that humans tend to react slowly or sometimes not at all. An automated system could speed reaction time, but we would have to secure the healing mechanism itself.

## 6 Conclusion

In this work, we have developed more realistic simulations that incorporate the dynamic nature of real-world systems and, hence, provide more realistic results. While we have focused on the LOCKSS protocol, many of our results can be extended to other peer-to-peer systems. Our data indicate that any peer-to-peer system in which the adversary can exploit multiple vulnerabilities will tend to break down. In LOCKSS, the average reference list corruption skyrockets, allowing the stealth-modification adversary to dominate the system. Furthermore, our experiments with the system-administrator model indicate that users of a peer-to-peer network must concern themselves not only with the security of their own system but also with the security of the other computers in the network. In a peer-to-peer network, no peer is an island.

The concept of using automated patch and/or repair systems also offers significant security benefits for peer-to-peer systems. While models of human behavior are necessarily inexact, our rough model for system-administrator responsibility indicates that patches and repairs need additional automation to ensure widespread distribution. As one possibility, the Ripple Healing technique draws on the same viral infection pattern employed to attack the network and uses it to drastically improve the security of the system. Such distributed repair techniques fit naturally with the peer-to-peer philosophy.

## Acknowledgments

The authors would like to thank the other members of the Harvard LOCKSS team: Geoff Goodell, Rachel Greenstadt,

Ian Becker, and Charles Duan. They also thank TJ Giuli for answering our questions about the LOCKSS code, David Rosenthal for suggesting various rates of infection and recovery to try, and Margo Seltzer and Michael Mitzenmacher for their feedback and suggestions for improving this paper. The EECS staff at Harvard provided assistance with hardware and software. They would also like to thank our anonymous reviewers for their helpful comments and suggestions. Finally, they thank Diana Seymour for her assistance, editing skills, and support.

## REFERENCES

[1] M. Abadi, M. Burrows, M. Manasse, and T. Wobber, "Moderately Hard, Memory-Bound Functions," *Proc. 10th Ann. Network and Distributed System Security Symp. (NDSS),* Feb. 2003.
[2] W.A. Arbaugh, W.L. Fithen, and J. McHugh, "Windows of Vulnerability: A Case Study Analysis," *Computer,* vol. 33, pp. 52-59, Dec. 2003.
[3] The Internet Archive, The Internet Archive Wayback Machine, http://www.archive.org/, 2004.
[4] Association of Research Libraries, ARL Statistics 2000-01, http://www.arl.org/stats/arlstat/01pub/intro.html, 2001.
[5] B. Boliek, "U.S. Music Industry Sues Song Swappers," http://www.reuters.co.uk/newsArticle.jhtml?type=internetNews&storyID=46%43035&section=news, 2004.
[6] P. Bungale, G. Goodell, and M. Roussopoulos, "Conservation vs. Consensus in Peer-to-Peer Preservation Systems," Technical Report TR-29-04, Harvard Univ., Nov. 2004.
[7] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," *Proc. OSDI: Symp. Operating Systems Design and Implementation,* 1999.
[8] M. Castro and B. Liskov, "Proactive Recovery in a Byzantine-Fault-Tolerant System," *Proc. Fourth Symp. Operating Systems Design and Implementation (OSDI),* Oct. 2000.
[9] J. Douceur, "The Sybil Attack," *Proc. IEEE Int'l Symp. Peer-to-Peer Systems,* Mar. 2002.
[10] Cooperative Association for Internet Data Analysis, "Telescope Analysis,"http://www.caida.org/analysis/security/telescope/, 2004.
[11] The Long Now Foundation The Rosetta Project, http://www.rosettaproject.org/, 2004.
[12] T.J. Giuli and M. Baker, "Narses: A Scalable, Flow-Based Network Simulator," Technical Report arXiv:cs. PF/0211024, Computer Science Dept., Stanford Univ., Stanford, Calif., Nov. 2002.
[13] A. Goldberg and P.N. Yianilos, "Towards an Archival Intermemory," *Proc. IEEE Conf. Advances in Digital Libraries,* pp. 147-156 1998.
[14] R. Greenstadt, G. Goodell, I. Becker, and M. Roussopoulos, "Establishing a Web of Trust in Sampled Voting Systems," Technical Report TR-09-04, Harvard Univ., May 2004.
[15] T. Jefferson, "Thomas Jefferson to Ebenezer Hazard, Philadelphia, February 18, 1791," *Thomas Jefferson: Writings: Autobiography, Notes on the State of Virginia, Public and Private Papers, Addresses, Letters,* 1984.
[16] J. Kubiatowicz, D. Bindel, Y. Chen, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "OceanStore: An Architecture for Global-Scale Persistent Storage," *Proc. ACM ASPLOS Conf.,* Nov. 2000.
[17] D. Malkhi and M. Reiter, "Byzantine Quorum Systems," *The J. Distributed Computing,* vol. 11, no. 4, pp. 203-213, Oct. 1998.
[18] P. Maniatis, M. Roussopoulos, T.J. Giuli, D.S.H. Rosenthal, M. Baker, and Y. Muliadi, "Preserving Peer Replicas By Rate-Limited Sampled Voting," *Proc. 19th ACM Symp. Operating Systems Principles,* pp. 44-59, Oct. 2003.
[19] P. Maniatis, M. Roussopoulos, T.J. Giuli, D.S.H. Rosenthal, M. Baker, and Y. Muliadi, "Preserving Peer Replicas By Rate-Limited Sampled Voting in LOCKSS," Technical Report arXiv:cs. CR/0303026, Stanford Univ., Mar. 2003.
[20] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Code-Red: A Case Study on the Spread and Victims of an Internet Worm," *Proc. Internet Measurement Workshop,* Sept. 2002.
[21] OpenBSD, OpenBSD Site, http://www.openbsd.org/, 2004.
[22] G. Orwell, *1984.* New York: New Am. Library, 1977.
[23] Honeynet Project, "Know Your Enemy: GenII Honeynets," http://project.honeynet.org/papers/gen2/, 2004.
[24] LOCKSS Project, LOCKSS Home Page, http://lockss.stanford.edu/, 2004.
[25] E. Rescorla, "Security Holes. . . Who cares?" *Proc. 12th USENIX Security Symp.,* pp. 75-90, Aug. 2003.
[26] D.S.H. Rosenthal, "LOCKSS Security," http://lockss.stanford.edu/lockssecurity.html, 2004.
[27] D.S.H. Rosenthal, P. Maniatis, M. Roussopoulos, T.J. Giuli, and M. Baker, "Notes on the Design of an Internet Adversary," *Proc. Adaptive and Resilient Computing Security Workshop,* Nov. 2003.
[28] A. Rowstron and P. Druschel, "Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility," *Proc. 18th ACM Symp. Operating Systems Principles,* Oct. 2001.
[29] E. Sit and R. Morris, "Security Considerations for Peer-to-Peer Distributed Hash Tables," *Proc. Int'l Workshop Peer-to-Peer Systems,* Mar. 2002.
[30] The Memory Hole, "Reasons Not to Invade Iraq," http://www.thememoryhole.org/mil/bushsr-iraq.htm, 2004.
[31] D.S. Wallach, "A Survey of Peer-to-Peer Security Issues," *Proc. Int'l Symp. Software Security,* Nov. 2002.
[32] Y. Wang, D. Chakrabarti, C. Wang, and C. Faloutsos, "Epidemic Spreading in Real Networks: An Eigenvalue Viewpoint," *Proc. IEEE Int'l Symp. Reliable Distributed Systems,* Oct. 2003.

**Bryan Parno** received the bachelor's degree in computer science from Harvard University in 2004. He is a PhD student in the Electrical and Computer Engineering Department at Carnegie Mellon University. With his advisor, Adrian Perrig, he studies network security and security for sensor and vehicular networks.

**Mema Roussopoulos** received the bachelor's degree in computer science from the University of Maryland at College Park, and the master's and PhD degrees in computer science from Stanford. She is an assistant professor of computer science on the Gordon McKay Endowment at Harvard University. Before joining Harvard, she was a postdoctoral fellow in the MosquitoNet Group at Stanford University. Her interests are in the areas of distributed systems, networking, mobile computing, and digital preservation. She is a member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.