How to Subvert LOCKSS

and What the LOCKSSmith Can Do About It

A Thesis presented

by

Bryan Parno

to

Computer Science

in partial fulfillment of the honors requirements

for the degree of

Bachelor of Arts

Harvard College

Cambridge, Massachusetts

April 6, 2004

Abstract

The LOCKSS (Lots Of Copies Keep Stuff Safe) project allows libraries to store and preserve electronic journals and other archival information through a system of inexpensive computers arranged in an ad-hoc peer-to-peer network. We develop a more accurate view of how the system will perform over time by simulating the system's behavior using a dynamic model in which peers can be subverted and repaired. This reveals certain systemic vulnerabilities not apparent in our static simulations, so we propose and evaluate countermeasures. One technique, Ripple Healing, performs remarkably well. We also propose and evaluate an alternate model based on the system administrators in the system. Finally, we develop a mathematical model of the stealth-modification adversary's attempts to modify system content while avoiding detection. This model allows us to improve our predictions of his behavior and analyze methods for thwarting his success.

Contents

List of Figures

Chapter 1

Introduction

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) spread 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 of the Internet, combined with the distributed trust system (which typically involves trusting a large majority of the other participants, rather than one central organization) of peer-to-peer systems, often leaves such networks vulnerable to subversion from within. This class of problems encompasses everything from an adversary actively subverting otherwise legitimate peers in the system 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 [8]. Thus, as development in the realm of peer-to-peer systems continues, it is important to consider the unique threats posed to such systems.

1.2 LOCKSS: A Digital Preservation System

The LOCKSS system [16] attempts to harness peer-to-peer's decentralized security benefits, 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. In other words, the libraries pay the publisher for access to the material, rather than for 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 already exists and has occurred in the real world.

In its March 2^{nd}, 1998 issue, *Time* magazine published an essay by George Bush Sr. and Brent Scowcroft describing the reasoning behind the United States' decision not to remove Saddam Hussein from power in Iraq during the first Gulf War. Among their reasons, they mention the enormous cost, the lack of multilateral support, and the likelihood that Iraqis would view American troops as occupiers - all of which

remain applicable today. While the article originally appeared on *Time*'s website 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 [31], 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" [13]. 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], LOCKSS operates on the assumption that the system will run on cheap PC's, making luxuries such as large RAID arrays and redundant power supplies unavailable. 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 and instead 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 in the system.

1.3 Know Thine Enemy

Following Sun-Tzu's admonition [29], this paper approaches the examination of the LOCKSS stealth-modification adversary from two complementary directions. First, we look at the development of increasingly sophisticated simulations that incorporate the possibility of multiple bugs and/or vulnerabilities in the system, as well as human factors that influence the rate at which peers are repaired, particularly the system administrator's level of responsibility. These new simulations more accurately reflect the behavior of a system in the real world. Then, we take the opposite approach and develop a purely mathematical model of the adversary's infiltration of the system. Both approaches offer a better understanding of how LOCKSS will perform over the long term and provide us with stronger mechanisms to ensure its continued viability.

Chapter 2

Background

2.1 LOCKSS Details

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 separate AUs. Each library in the LOCKSS system is assumed to begin with a list of "friends" also using 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, i.e. 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 a larger subset of the population. The peer initializes its reference list with all of its friends, and then periodically (currently every three months) conducts an "opinion poll" by sending invitations to a subset of its reference list. This subset is called the inner circle (see

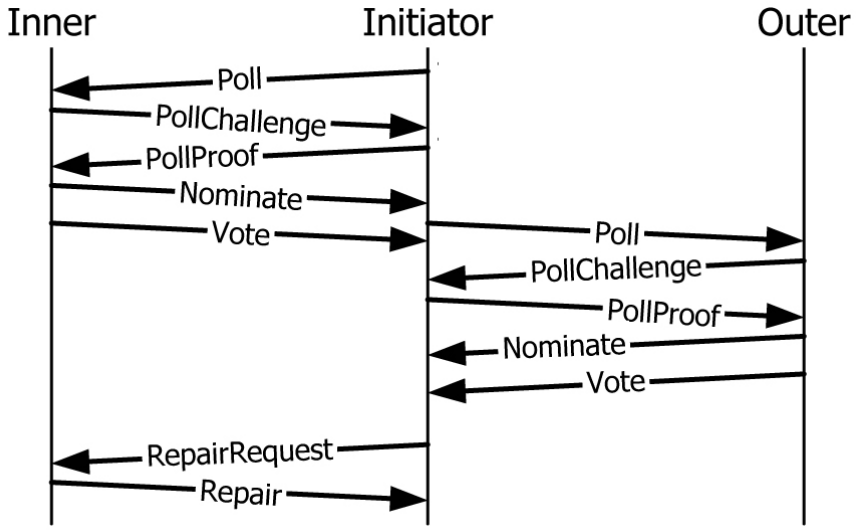

Figure 2.1: **Voting Protocol** *This figure shows the messages exchanged between LOCKSS peers participating in an opinion poll. The left side represents an inner-circle peer, and the right side represents an outer-circle peer. Time flows from top to bottom.*

Figure 2.1). When a peer receives a poll invitation, it responds with a Poll Challenge message, asking the initiator for a proof of effort based on the Poll Challenge in the message. 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. While the peer participates in this poll (which currently takes approximately five hours), it ignores all other incoming poll requests.

After the initiator has successfully responded to the challenge, the invited peers send a vote in the form of a secure hash of the AU back to the initiator. Each 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. We use the outer circle for discovery purposes, i.e. to expand our list of known

peers in the system.

Once the initiator has received all of the votes (and assuming it has received enough to reach a quorum), it compare each vote with a hash of its own AU. If an overwhelming number (set as a system parameter[1]) of votes agree with its copy, it assumes the copy remains undamaged, so it 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 inner-circle peers that disagreed with its copy. The disagreeing 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, i.e. illegitimately requesting copies of AU's that one does not own, 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 and 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 successful poll, the initiator updates its reference list to avoid relying on any one set of peers. First, it removes any disagreeing inner-circle peers from the reference list, as well as randomly removing enough agreeing inner-circle peers to bring the total removed up to Q, the number needed to form a quorum. Second, it inserts all of the outer-circle peers that agreed with the vote into the reference list. Finally, it inserts a small, randomly chosen subset of peers from its friends list. We call this operation "churning". The first two steps prevent the long-term accumulation of reputation. This prevents an adversary from agreeing in a few polls in order to

[1]Currently, the system uses 70% as the threshold for an "overwhelming" vote.

worm its way into the reference list, only to cash in by attacking. The final step, churning, helps slow the growth of the adversary's presence in the reference list, since the friends list tends to remain less corrupt than the general 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.

2.2 Potential Adversaries

The design of LOCKSS inherently necessitates the ability to defend against extremely powerful, patient adversaries attempting to subvert the system. LOCKSS must preserve data for decades, and it takes little imagination to envision potential attackers. As Orwell notes, "Who controls the past controls the future" [19]. For example, a tobacco company might want to alter the results of a study linking smoking with lung cancer, or a certain Redmond-based company might wish to eliminate a pesky 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 [25]. 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 Chapter 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. 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 counter-measures, 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 to 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 lurking adversary that truly undermines the entire motivation for 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 [16]. Later, we will explore the effects of altering this assumption.

Chapter 3

Related Work

The original LOCKSS team at Stanford currently supports the existing deployment of LOCKSS at over 80 institutions worldwide, with the support of more than 50 publishers representing over 1,000 titles [21]. They are also investigating stronger measures to combat the attrition adversary, using effort-balancing and admission-control techniques. At Harvard, Becker, Goodell and Greenstadt are investigating the security model of the system, analyzing the tradeoffs involved in adding public key based authentication to the LOCKSS protocol.

In the peer-to-peer realm, systems such as PAST [26], OceanStore [14] and Intermemory [12] 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. Other papers in the peer-to-peer realm often follow a similar pattern - if they mention security at all, it tends to come as an afterthought with a purely qualitative analysis of the system (e.g. Chord [28], CAN [22] and semantic overlay networks [30]).

Several papers provide a general survey of security issues facing peer-to-peer systems. Wallach provides an overview of such issues [32], with a focus on routing and file sharing. Morris and Sit offer a more in-depth exploration of these issues [27], 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 [33]. 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 over 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

magazines and journals of greatest interest to scholars. Finally, the Wayback Machine is inherently a centralized process, the direct opposite of LOCKSS' decentralized approach.

In a process reminiscent of Byzantine fault-tolerance schemes (e.g. [6], [7], and [15]), 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.

Approaching the problem from a hardware perspective, the Rosetta Project [10] is creating a 1,000 language corpus and using a micro-etching 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. As a more mundane approach, 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.

Chapter 4

Simulating Dynamic Populations

4.1 Motivation

Thus far the LOCKSS simulations have assumed that the adversary starts with a certain proportion of compromised peers and that these peers remain subverted throughout the simulation [16]. 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," "malign" or "bad." 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," "good" or "loyal." 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 [11]. 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. In the simulations, we use a population of 1000 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%. In each simulation, the adversary begins with some percentage of the peers under his control. We also give the adversary additional identities that can masquerade as legitimate peers, allowing us to model a Sybil attack. While these attacks generally spawn thousands of IDs, the LOCKSS system makes this much more difficult, since each ID must perform a massive amount of computation to prove its interest in polls. This places a computational bound on the number of useful IDs the adversary can create. We set the number of extra identities to 200, recalling our assumption of a powerful adversary.

The simulations run for 20 years (7200 days), and the results represent the average of 10 simulations using different random seeds. Standard deviations are less than 2%. We present the reference list corruption as the percentage of a healthy peer's reference list composed of subverted peers. Several graphs present an average result from the static simulations as a reference point. The static simulation has a constant subversion level of 30%.

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

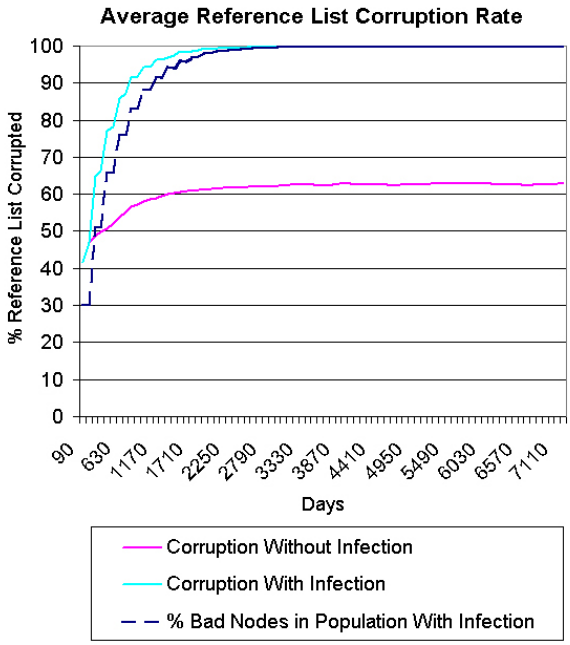

Figure 4.1: **The Effect of Infection** *In this simulation, the adversary exploits a new vulnerability that affects 30% of the unsubverted population every six months (182 days). One sample from the static simulations is shown for comparison. The solid lines indicate reference list corruption and the dashed line indicates the percentage of subverted peers in the system. We will follow this convention in subsequent graphs as well.*

install in more than 7 years" [18], 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. However, in general, one would expect to see the same general behavior shown in Figure 4.1. This simulation was run with an initial population of 1000 peers, with 30% initially subverted and a churn rate of 10%. Infections occur twice a year and affect 30% of the unsubverted[1] population. In the figure, the normal reference list corruption level plateaus around

[1]As an alternative, we could assume that the exploit affects 30% 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.

63%. 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 malign peers always recommend other malign peers, while loyal peers recommend a mixture of malign and loyal peers, the overall corruption of the reference lists should increase even faster than the number of malign 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. Nonetheless, we may find it useful to know just how fast the number of subverted peers in the system will grow. If we assume that a given vulnerability affects a fixed percentage, *virulence*, of the unsubverted peers, then we can derive an expected growth rate for the number of subverted peers M_t in the system after t infections using the recurrence relation

$$M_{t+1} = M_t + virulence * (P - M_t) \tag{4.1}$$

$$= (1 - virulence) * M_t + virulence * P \tag{4.2}$$

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

$$f(t+1) = af(t) + k \tag{4.3}$$

has a solution of

$$f(t) = (f(0) - \frac{k}{1-a}) * a^t + \frac{k}{1-a} \tag{4.4}$$

we expect the growth of subverted peers to follow the formula:

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

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 the speed and extent of Internet viruses and worms or the rate at which systems are patched and repaired. Work such as the Network Telescope [9] maintained by CAIDA (Cooperative Association for Internet Data Analysis) and the Honeynet Project [20] provide some hints of what we can expect to see in the future. A Network Telescope consists of a portion of routed IP space that does not expect legitimate traffic. Monitoring this space for unexpected traffic can indicate a network attack in progress or the beginnings of a new worm. 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.

Based on data collected at CAIDA, Moore et al. determined that Internet users responded surprisingly slowly to the Code Red threat [17]. 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 [23] of the response to the OpenSSL remote buffer overflow exploit announced in July 2002. Although users running OpenSSL tend to be more security-conscious and overwhelmingly run some variant of Unix, his data show a sluggish response rate very similar to that for the Code Red vulnerability.

Based on this data, we developed a basic exponential decay function of the form

$$Percent\_Compromised\_Peers = A * e^{-B*days\_elapsed} + C \tag{4.5}$$

$$Percent\_Healed\_Peers = 1 - Percent\_Compromised\_Peers \tag{4.6}$$

to model the rate at which machines are repaired. We consider both pessimistic and optimistic versions of the model. For our pessimistic model, the values of A, B, and C were chosen[2] such that 33% of the peers would remain vulnerable after a month, with this value tapering off to 20%, meaning that the system never fully recovers. For the more optimistic model, we chose constants[3] that would still leave 33% of peers vulnerable after one month but would eventually bring the level of vulnerability close to 0 (see Figure 4.2).

As shown in Figure 4.3, the selection of the healing model makes a significant

[2] $A = 80$, $B = 0.0606$, and $C = 20$

[3] $A = 99.99$, $B = 0.0369$, and $C = 0.01$

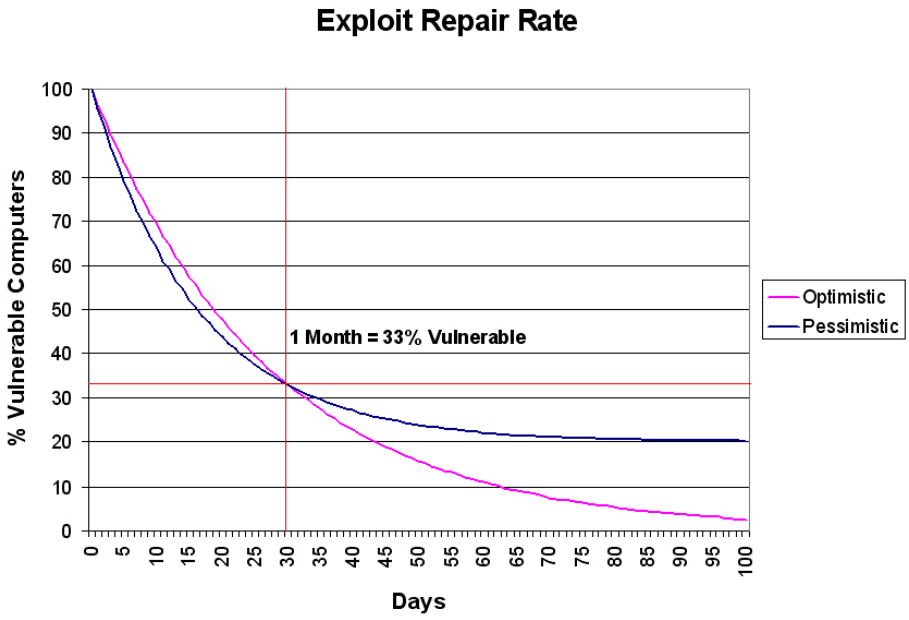

Figure 4.2: **Healing Rates** *Two possible models for the rate at which computer administrators patch/fix vulnerable computers.*

impact on the system's performance. With the pessimistic model, only 80% 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 (illustrated with dashed lines), and the quality of the reference lists degrades rapidly. With the optimistic model, the system can recover from most of the damage (approximately 99% of infected peers recover) and thus valiantly resists the adversary's encroachments. We assume 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, team members were able to patch 95% of the deployed systems to fix a (hypothetical) vulnerability within

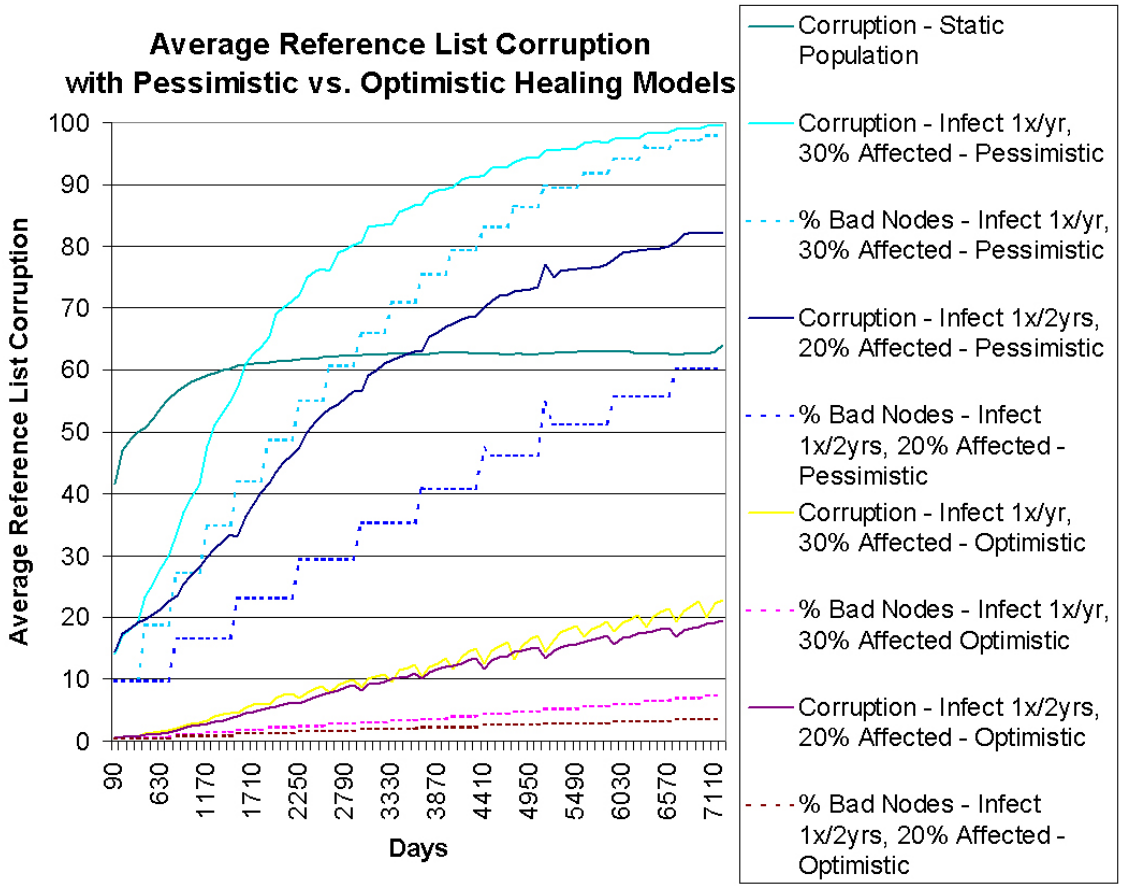

Figure 4.3: **Impact of Optimistic vs. Pessimistic Healing Models** *Using a healing model in which most users (approximately 99%) eventually patch their vulnerable system significantly improves the system's performance. Representative sample simulations shown.*

48 hours [24], indicating a pattern closer to the optimistic model. However, as the system grows and evolves, it will be difficult to maintain this efficiency. Also, examining the general trends in the reference list corruption levels for both the optimistic and the pessimistic models reveals 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 Repairing Recurring Infections

To examine the effectiveness of repairs in the face of repeated infections, we ran several simulations while varying the percentage of the population affected by each exploit (Figure 4.4), the rate at which exploits occur (Figure 4.5) and the initial level of systemic subversion (Figure 4.6). Since the authors of the original LOCKSS paper [16] note the importance of churning (adding peers from the friends list to the reference list), we also ran simulations to analyze the extent to which churning helped resist the effects of infection (Figure 4.8).

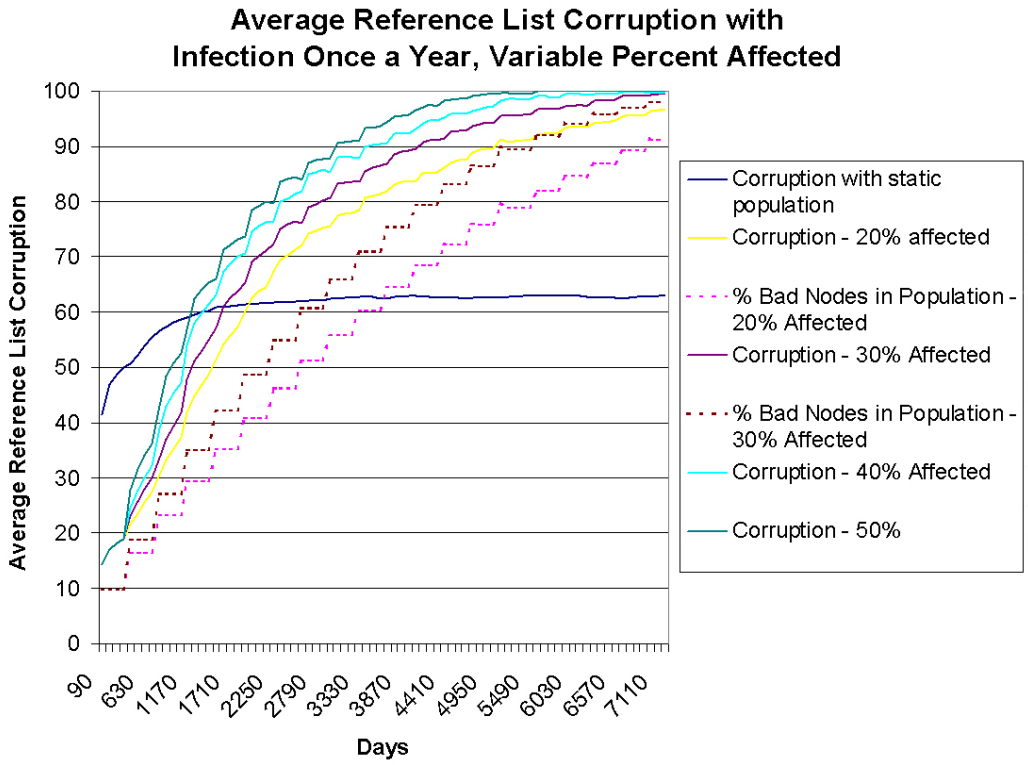

Figure 4.4: **Varying Exploit Impact** *Varying the percentage of peers affected by each exploit changes the behavior of the average reference list corruption of the good peers' reference lists. One sample from the static simulations is shown for comparison. The lines for the percentage of bad peers for 40% and 50% affected have been omitted for the sake of clarity, but they follow the same general trend as the 20% and 30% lines.*

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 Figure 4.4 were run with an initial subversion level of 30%, a 10% 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 bad peers (shown as dashed lines) 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 Figure 4.5 were run with an initial subversion level of 30%, a 10% churn rate, the pessimistic healing model and infections that affected 30% 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 (shown as dashed lines), grows in a stepwise fashion. While the frequency of infection clearly has an impact on the system, even the system with infections occurring only once every two years surpasses the static population's reference list corruption level after eight years, and by the end of the simulation (twenty years), the corruption has completely overwhelmed the system.

Interestingly, varying the level of subversion in the system has little impact on a dynamic population (see Figure 4.6). For these simulations we held the rate of infection and the percentage of the population affected constant (at once a year and 30% affected per exploit, respectively) and varied the initial subversion from 20%

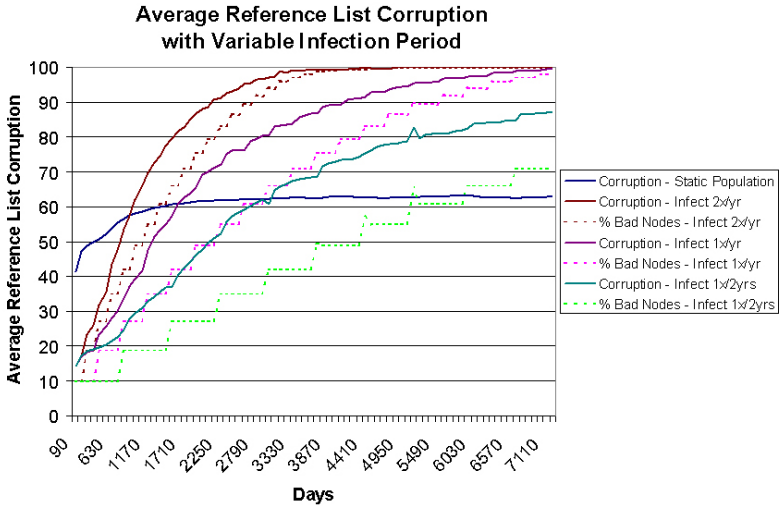

Figure 4.5: **Varying Infection Rate** *Decreasing the rate of infections decreases the average corruption of the reference lists, but the general trend towards universal reference list corruption remains. One sample from the static simulations is shown for comparison.*

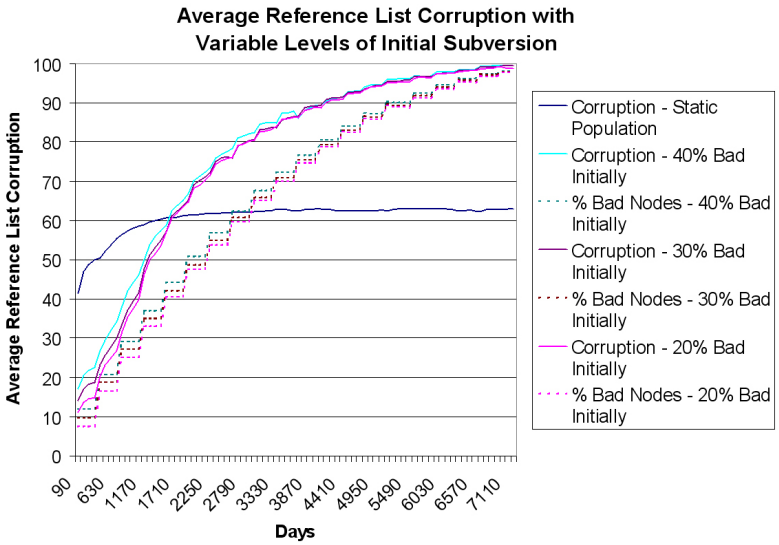

Figure 4.6: **Varying Initial Subversion** *Varying the initial subversion of the population has little effect on the overall growth of reference list corruption. Unfortunately, all of the dynamic simulations tend towards systemic reference list corruption. One sample from the static simulations is shown for comparison.*

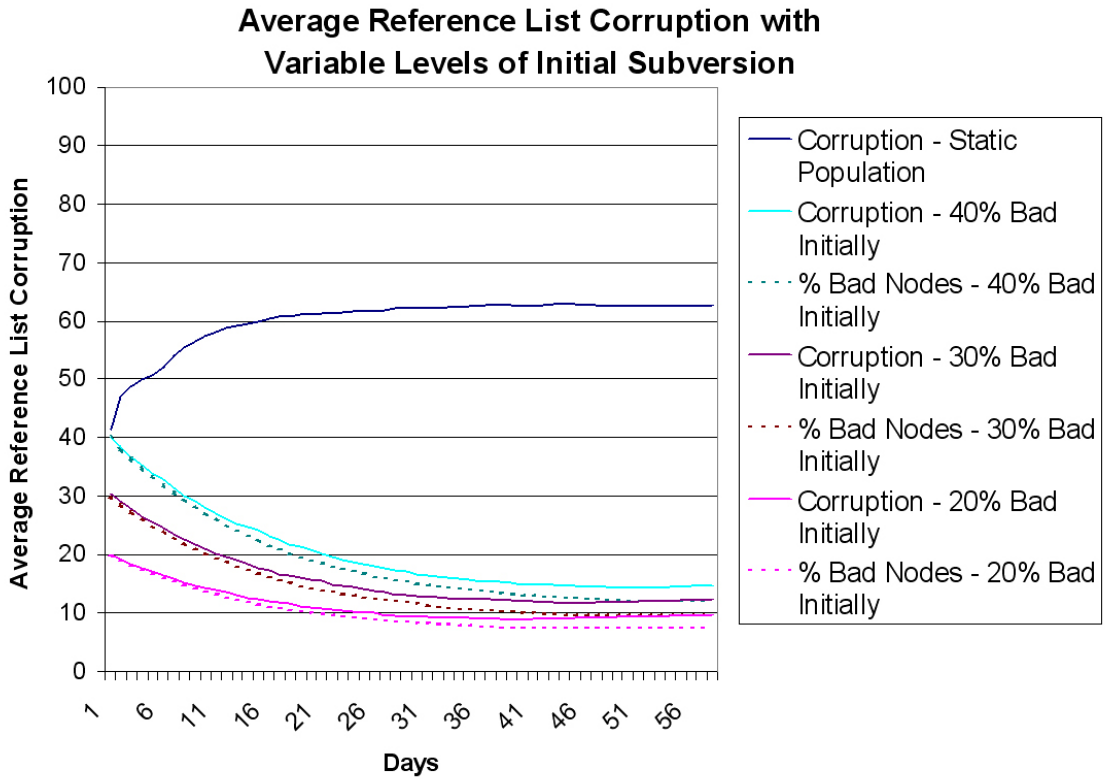

Figure 4.7: **Varying Initial Subversion - Zoomed In** *On a smaller time scale (the first 60 days of the simulation), we can see that despite varying levels of initial subversion, all of the simulations converge towards the same level of reference list corruption by the 60^{th} day.*

to 50% 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 Figure 4.7). Since the first new infection does not hit immediately (until the 365^{th} 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.

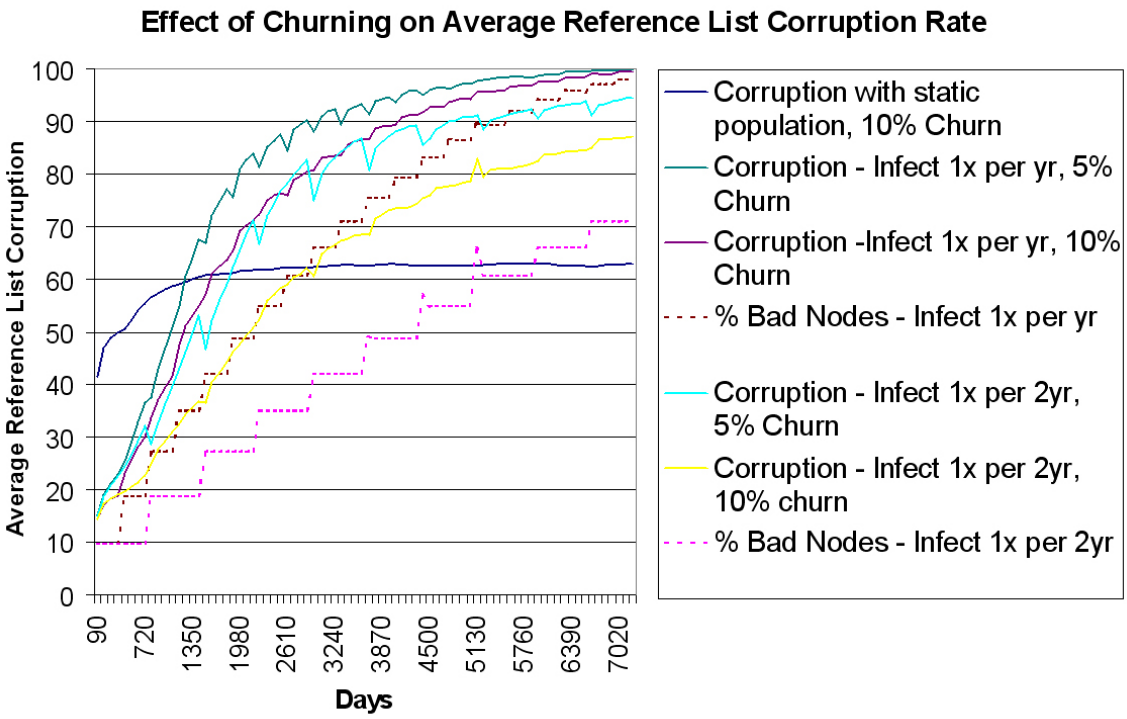

Figure 4.8: **Varying Churn Rate** *Increasing the percentage of peers churned into the reference list from the friends list reduces the average level of corruption in the reference list. One sample from the static simulations is shown for comparison.*

As shown in Figure 4.8, increasing the rate at which peers 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%, infections that affected 30% 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.

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 towards 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 Figure 4.9, Clean Start creates drastic variations in the level of reference list corruption in the system, as the healing effects battle with the repeated infections. In these simulations, we begin with 30% subversion and assume infections occur once a year and affect 30% of the population. All of the good peers use a 10% churn rate. Overall, the corruption rises much more slowly than 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 much 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.

4.6.2 Ripple Healing

The other modification we investigated, Ripple Healing, initially seems quite similar. Whenever a peer discovers that it has been compromised (using the normal discovery

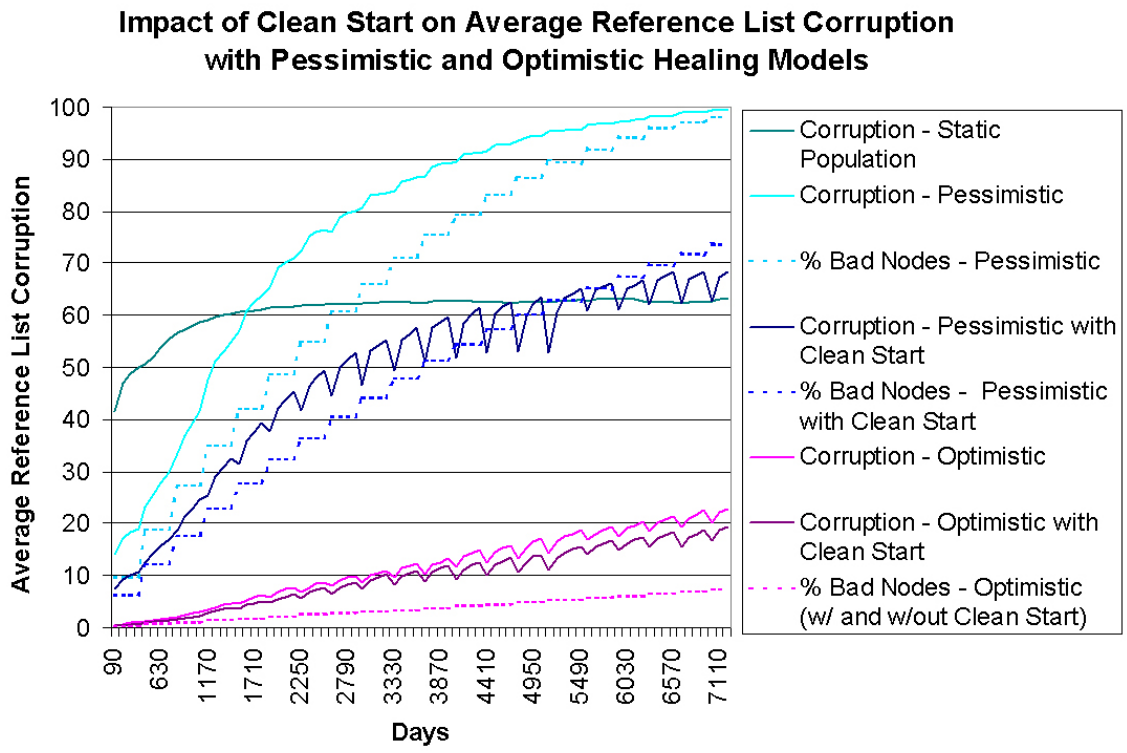

Figure 4.9: **Effect of the Clean Start Technique** *Giving each healed peer a purified reference list improves performance, despite oscillations. Simulations were run with 30% initial subversion and assumed that infections occurred once a year and affected 30% of the population. All of the good peers use a 10% churn rate.*

mechanism dictated by either the optimistic or the pessimistic healing model), it keeps the same reference list as before, but it adds all of the peers on its friends list to a quarantine list. The next day, we examine the list and heal any subverted peers, though they do not add their friends lists to the quarantine. This simulates a system administrator realizing his system has been compromised and alerting his friends that they should investigate their systems too. We omit a graph for this case, since even with 60% initial subversion and the pessimistic healing model, the Ripple Healing cured 100% 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 (as explored 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 a compromise 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.

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:

$$M_{t+1} = M_t - H * \frac{F * M_t}{P} = (1 - \frac{HF}{P})M_t \tag{4.7}$$

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 righthand side of Equation 4.7). Drawing on our work from Section 4.3, we know

that a recurrence relation of this form has the solution:

$$M_t = M_0(1 - \frac{HF}{P})^t \tag{4.8}$$

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

$$M_t = M_0 * e^{-\frac{HF}{P}t} \tag{4.9}$$

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 on the number of peers healed each day and the number of friends the average peer possesses. Figure 4.10 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 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.

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

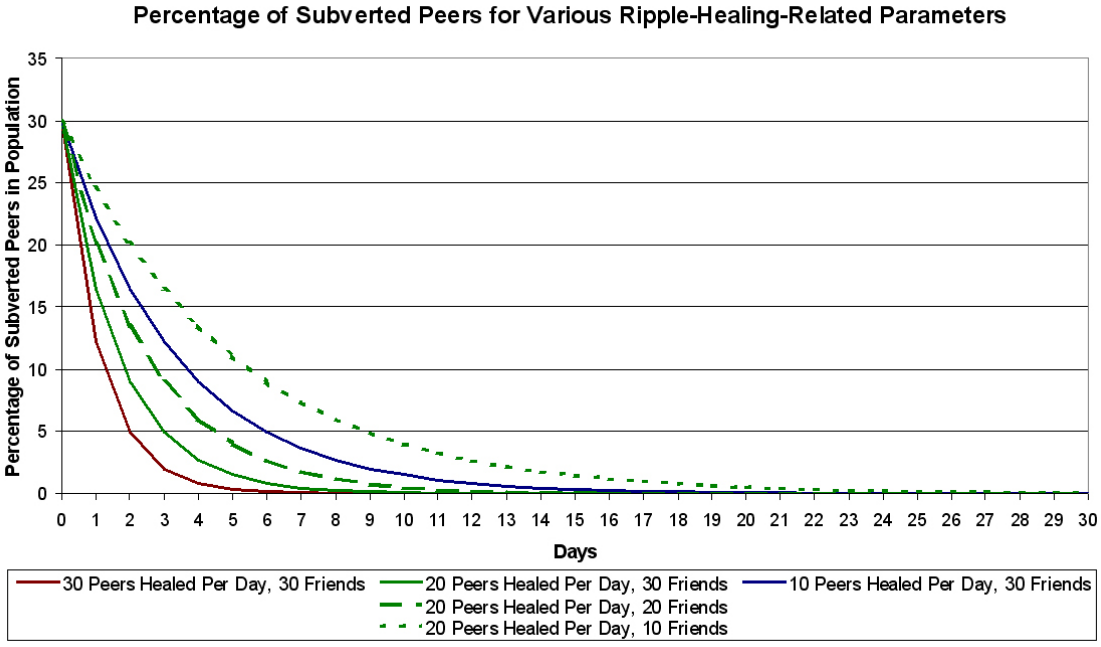

Figure 4.10: **Effect of Ripple Healing** *The impact of Ripple Healing depends on the number of friends each peer has, as well as the number of peers healed each day.*

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 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 [23] and Arbaugh [2] note, the vast majority of system administrators tend to fall into the latter rather than the former category. Systems remain unpatched for known vulnerabilities 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.

In our new model of the world, we assign each peer a system-administrator rating

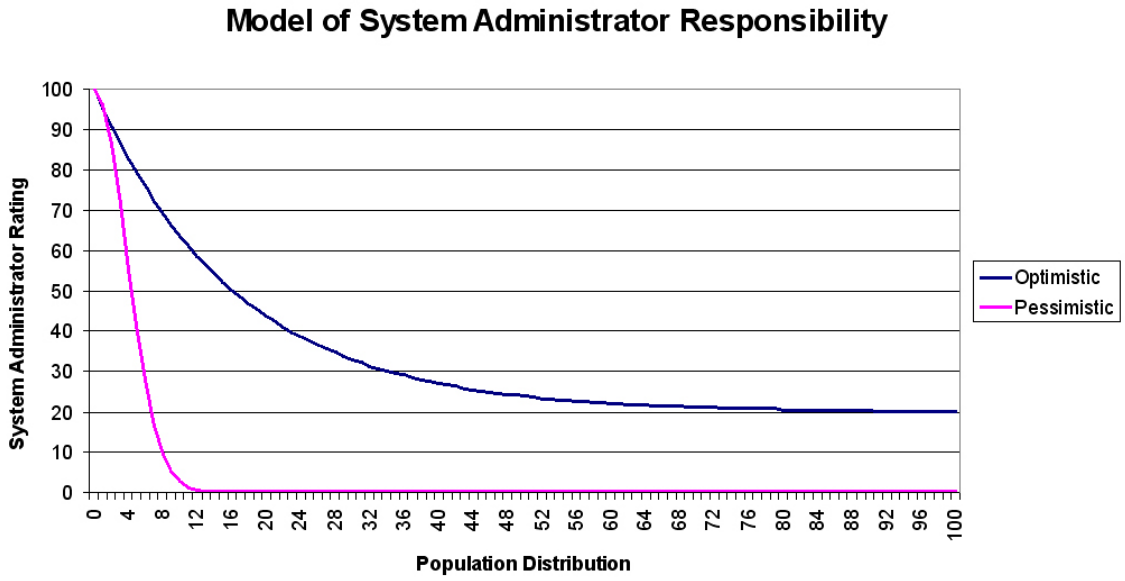

Figure 4.11: **System Administrator Abilities** *A few peers have highly skilled system administrators, but the majority have mediocre ratings. The x-axis shows the distribution of skill ratings, indicating for example that in the pessimistic model, approximately 9% of the population has a rating of 10 or above.*

(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 Figure 4.11). 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. Once again, we experimented with both an optimistic and a pessimistic model. To test the impact of these ratings, we repeated a selection of the previous experiments with both 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:

$$P_{discovery} = \frac{sys\_admin\_rating}{100} \tag{4.10}$$

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, since most peers have a rating of $0.274 \approx \frac{100}{365}$, indicating that most peers only heal 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:

$$P_{infection} = \frac{100 - sys\_admin\_rating}{100} \tag{4.11}$$

The graph in Figure 4.12 compares a sampling of the results. We ran the simulations with infections occurring once per year and affecting 30% of the population (in the simulations that did not use the system administrator's rating to determine infections). The system started with 30% of the population subverted and used a 10% churn rate. 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. Given that in this case even the worst system administrators heal their peer every five days, this does not seem unreasonable. The anomalous spikes that occur at certain days (e.g. Day 5850 and Day 6210) occur because we only sample the corruption every 90 days, and these data points happen to catch the system almost immediately after an infection, before the healing process has a chance to take effect. The graph in Figure 4.13 illustrates the correspondence between the spikes in the corruption rate and the proximity of an infection for a simulation run using the pessimistic model.

However, while the reference list corruption level with optimistic healing remains

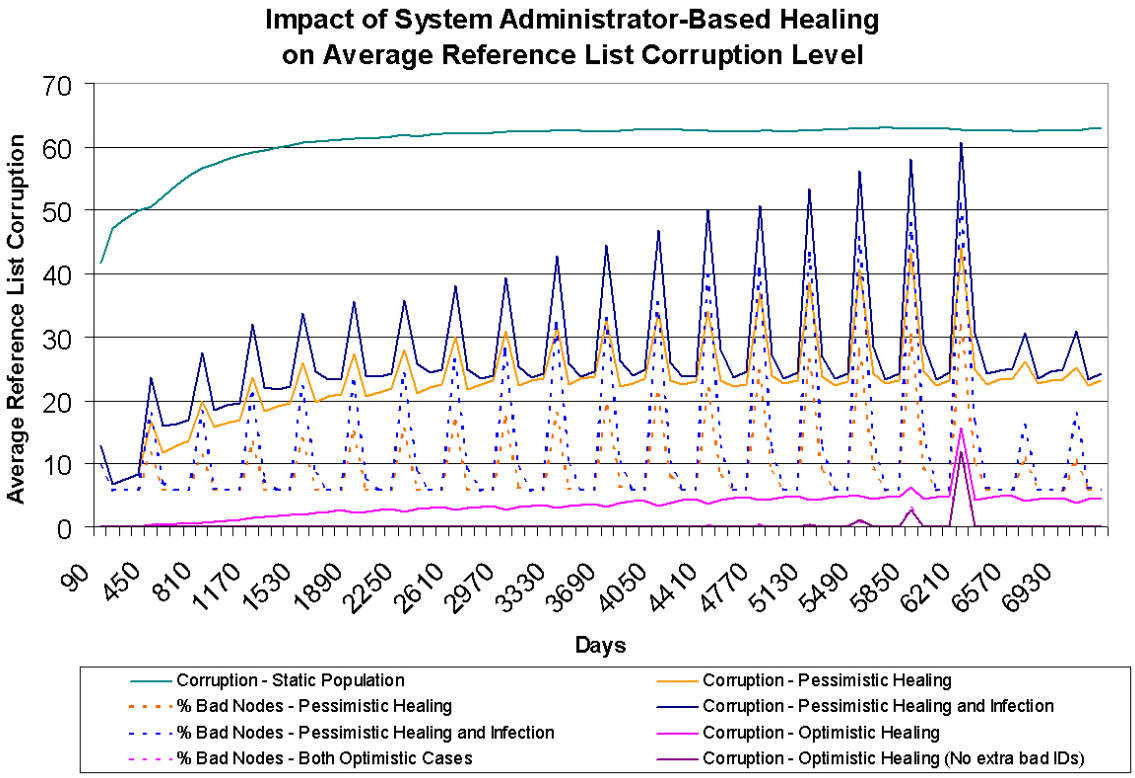

Figure 4.12: **Infection and Healing Based on System Administrator Abilities** *This graph illustrates the effects of the optimistic and pessimistic system-administrator-based infection and healing models.*

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. Remember that in addition to starting with control of a portion of the peers in the population (30% in this case), we also give the adversary an additional 200 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 does indeed go to zero (except for the extraneous spikes near an infection). This emphasizes the importance of designing a system to defend against a Sybil attack, since even if the adversary

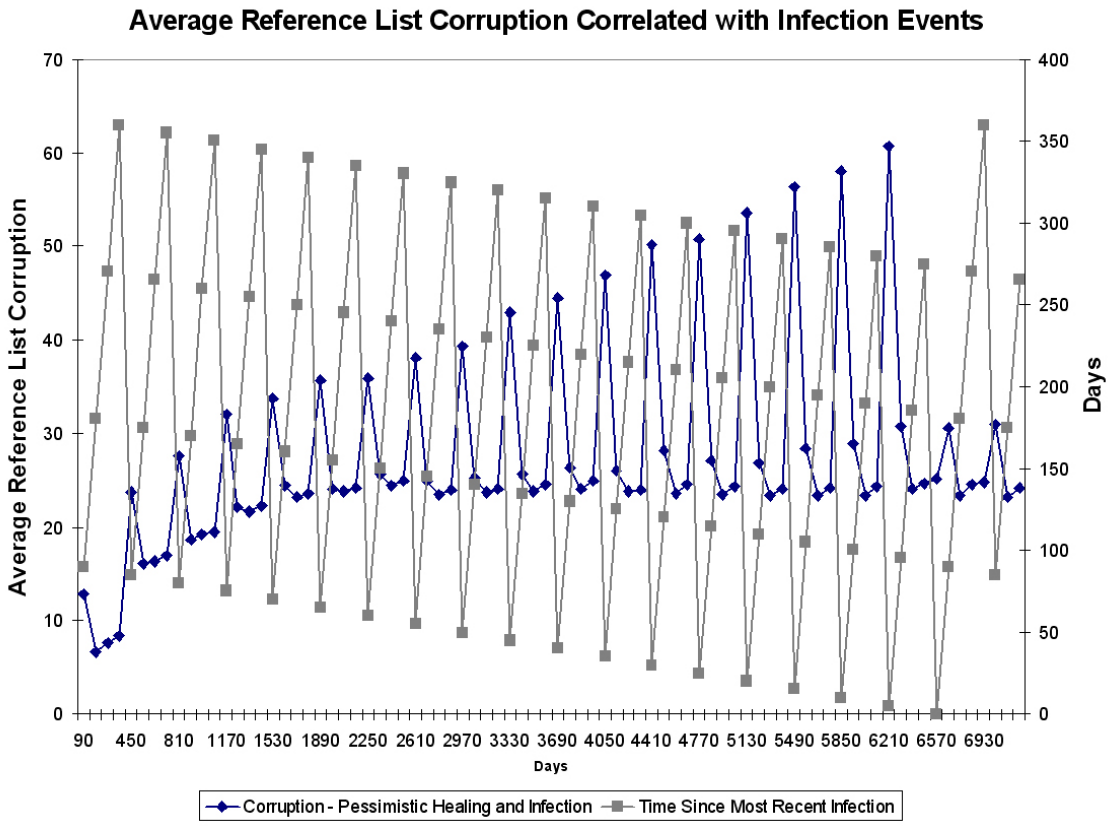

Figure 4.13: **Infection Plotted with Reference List Corruption** *This graph charts the average corruption of the reference lists along with the number of days that had passed since the most recent infection when the reference list corruption level was sampled (plotted on the righthand y-axis). Dips in the time since an infection tend to correspond with spikes in the level of reference list corruption.*

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

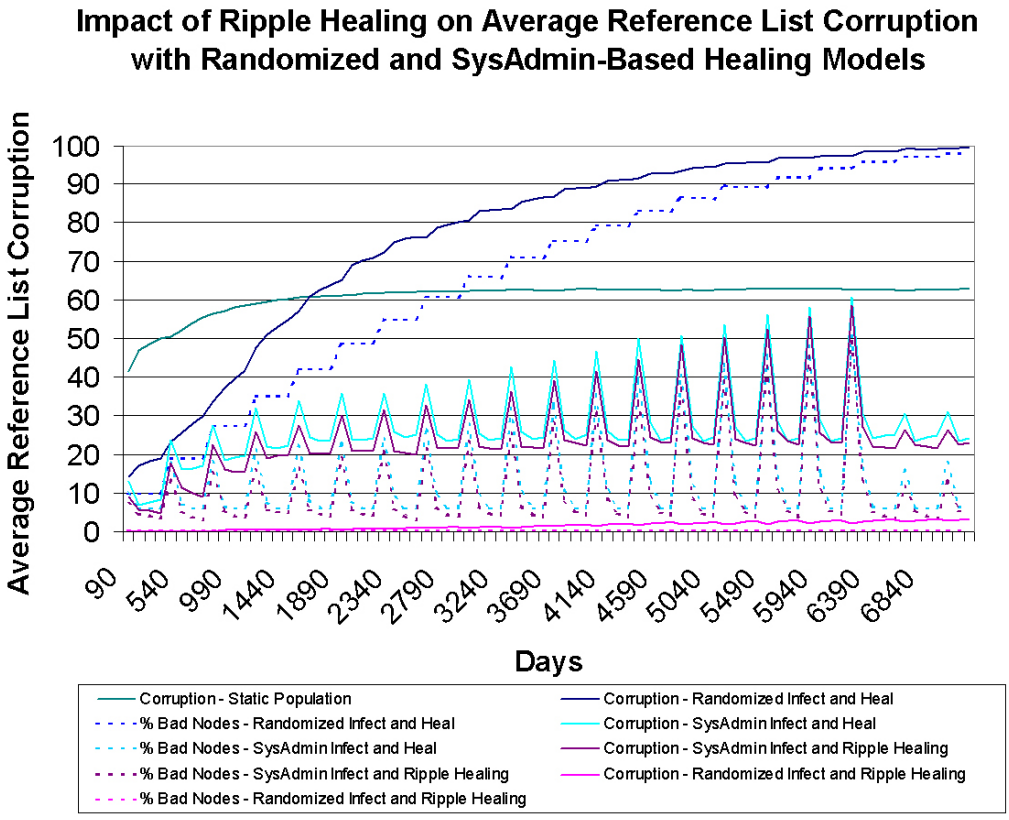

Figure 4.14: **Effects of Ripple Healing** *Ripple Healing provides only a marginal improvement using the system-administrator model, unlike the huge gains it gives the randomized model. In the simulations shown here, we illustrate the difference between using the randomized infection/healing model with and without Ripple Healing, and using the system-administrator model with and without Ripple Healing.*

randomized 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 4.6), 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 the 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 model (see Figure 4.14). Indeed, in the system-administrator model, the Ripple Healing technique provides a marginal improvement at best. In the randomized system, every peer responded 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 heal themselves are also those least likely to respond to an alert.

4.8 Implications

The results 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 decisively influences 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. (Figure 4.15 shows the average corruption of the reference lists for varying rates of infection in both models. We began with 30% of the population subverted and used a 10% churn rate. In the simulations using the randomized model, each infection affected 30% 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 performs significantly worse in the system-administrator model, for the reasons given above. Thus, selecting an appropriate model for the system will help determine both the expected behavior of the system and the techniques that may successfully combat reference list corruption within the system. 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 him- 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 that have successfully patched the vulnerability.

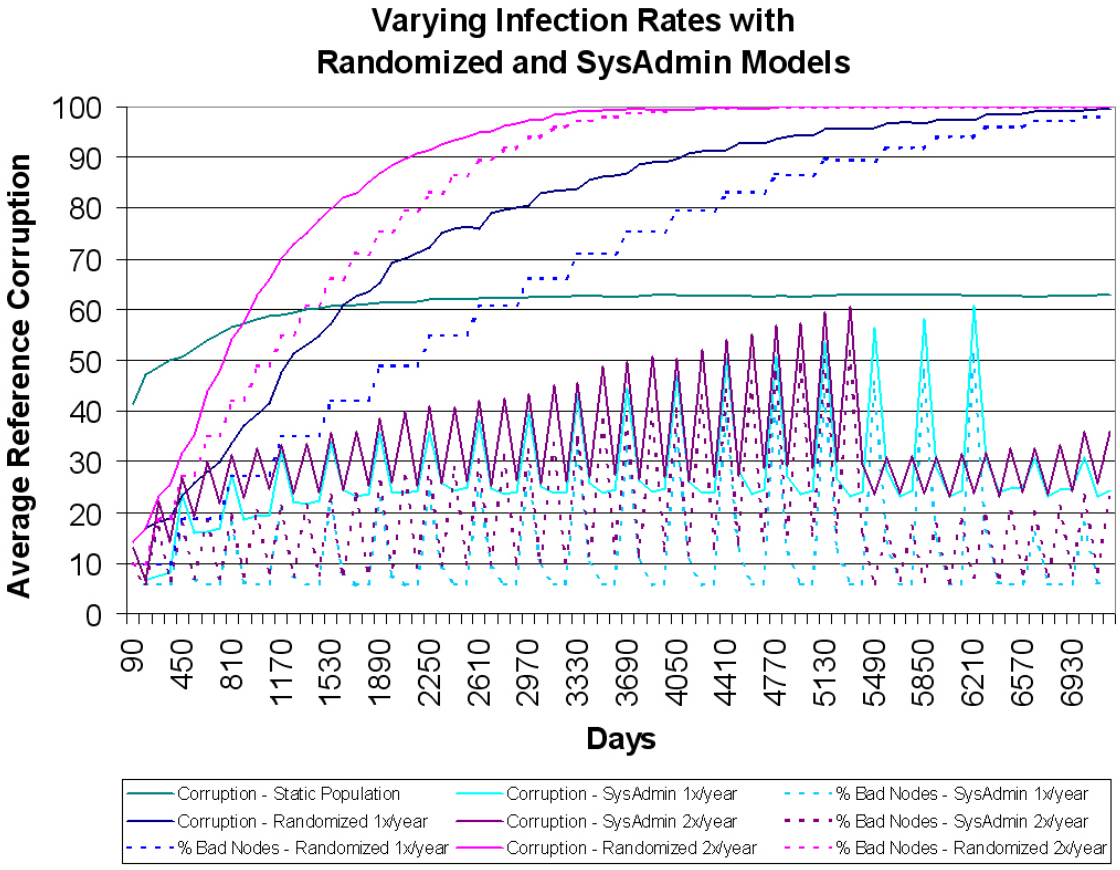

Figure 4.15: **Randomized vs. System Administrator Systems** *A comparison of the randomized and system-administrator models for various rates of infection. In general, the system-administrator model demonstrates better performance (i.e. less average reference list corruption). In the legend, the entries 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% of the unsubverted peers.*

Chapter 5

A Mathematical Model

5.1 Motivation

Thus far, the vast majority of the work examining potential LOCKSS adversaries relies on simulation results. While these results certainly offer hope that the LOCKSS protocol will remain secure, this approach has several drawbacks. Aside from the time and computational effort expended on each simulation, the accuracy of the results depends on a faithful representation of the protocol in code and offers little in the way of guarantees. A mathematical model, on the other hand, allows us to make stronger security guarantees. It also facilitates an analysis of exactly which variables factor into the adversary's success, allowing us to develop stronger safeguards against systemic damage by changing system parameters or adjusting the voting protocol. As a motivating example, we have developed a model of the lurking adversary that attempts to predict the growth of corruption in the reference list, allowing us to determine the probability that the adversary will cause permanent damage to the system. For now, we limit ourselves once again to a static population, leaving aside the complications introduced by a dynamic population.

5.2 Variable Definitions

We begin by defining the variables necessary for a rough view of the system.

- C = churn rate
- Q = number of inner-circle votes needed for a quorum
- L_{ic} = number of loyal inner-circle peers
- M_{ic} = number of malign inner-circle peers
- N = number of peers in the inner circle
- V = number of inner-circle peers that respond to a poll invitation
- T = target reference list size (currently $3 * N$ - this gives the peer a buffer, so that the reference list rarely becomes depleted)
- X = number of peers in the outer circle
- P = total number of peers in the population
- M_0 = the initial number of malign peers in the population
- F = size of friends list
- M_F = number of malign peers in the friends list
- M_{oc} = number of malign peers in the outer circle
- R_t = size of reference list at time t
- M_{rt} = number of malign peers in the reference list at time t

5.3 Analysis

Given M_0, P, F, C, etc., we want to find a general solution for M_{rt}.

5.3.1 Setup

As initial conditions, we know that the initial corruption of both the reference list and the friends list is proportional to the subversion of the population as a whole.

$$M_{r0} = \frac{M_0}{P} R_0 \tag{5.1}$$

$$M_F = \frac{M_0}{P} F \tag{5.2}$$

5.3.2 Calculation

Intermediate Steps

Inner-Circle Corruption

Assume that at some time t, we know the average size of the reference lists, R_t, and the current level of corruption in the reference lists, M_{rt}. We would like to calculate these statistics at the next unit of time, i.e. we would like to find R_{t+1} and $M_{r(t+1)}$. Assuming we choose inner-circle peers from the reference list at random, we have:

$$M_{ic} = \frac{M_{rt}}{R_t} N \tag{5.3}$$

$$L_{ic} = N - M_{ic} \tag{5.4}$$

Outer-Circle Nominations

When we request outer-circle nominations, we will assume:

1. All peers send agreeing votes
2. Each malign peer will only nominate other malign peers
3. All reference lists are corrupted at the same rate

Based on these assumptions, both the malign and the loyal peers will contribute to the number of malign peers in the outer circle. We want to invite enough peers, X, to achieve our target reference list size, i.e. $R_t + X = T \Rightarrow X = T - R_t$. To account for the Q peers that we remove from our list at the end of the round, we will actually set $X = target - R_t + Q$. We choose $target$ to account for the peers that we will churn in from our friends list, so $target = \frac{T}{(1+C)}$. The X outer-circle peers will be chosen evenly and at random from the various inner-circle nominations, so each inner-circle peer will contribute $\frac{X}{N}$ peers to the outer circle. Since we assume each malign peer will recommend only malign peers, the malign peers in the inner circle will contribute $M_{ic} * \frac{X}{N}$ malign nominations. A loyal inner-circle peer will also (inadvertently) contribute some number of malicious peers. If we assume that the proportion of malicious peers in its reference list is also $\frac{M_{rt}}{R_t}$, then the loyal inner-circle peers will collectively nominate $L_{ic} * \frac{X}{N} * \frac{M_{rt}}{R_t}$.

Thus, the number of malicious peers in the outer circle will be:

$$M_{oc} = M_{ic} * \frac{X}{N} + L_{ic} * \frac{X}{N} * \frac{M_{rt}}{R_t} \tag{5.5}$$

Reference List Updates

We have assumed that everyone cooperates and agrees with our copy of the AU, meaning $V = N$. Let us further assume that there are no collisions, i.e. we never attempt to insert a peer into our reference list if it is already present. We perform three updates to the reference list:

1. Remove a random Q peers (in the protocol, we remove $(Q - \#disagreeing)$ peers, but we have assumed everyone agrees with us, so $\#disagreeing = 0$)
2. Add all outer-circle peers that agreed (in this case, all X of them)
3. Insert $C * R_t$ peers from the friends list (churn the reference list)

We can now look at the effect these updates have on the number of malign peers in our reference list:

1. Remove $\frac{M_{ic}}{N} * Q$ malign peers
2. Add M_{oc} malign peers
3. Add $C * R_t * \frac{M_F}{F}$ malign peers

Combining these effects, we should have

$$M_{r(t+1)} = M_{rt} - \frac{M_{ic}}{N} * Q + M_{oc} + C * R_t * \frac{M_F}{F} \tag{5.6}$$

malign peers in our reference list. This gives us an average reference list corruption level of $\frac{M_{r(t+1)}}{R_{t+1}}$, where R_{t+1} is:

$$R_{t+1} = R_t - Q + X + C * R_t \tag{5.7}$$

Granted, these are recurrence relations, but even in this form they can be useful, and if necessary, they can be solved to provide a closed-form solution. For example, if we recall that $X = \frac{T}{1+C} - R_t + Q$, then we can rewrite Equation 5.7 as:

$$R_{t+1} = C * R_t + \frac{T}{1+C} \tag{5.8}$$

Again, drawing on our work in Section 4.3, we can eliminate the recurrence and express R_t as:

$$R_t = (F - \frac{T}{1-C^2}) * C^t + \frac{T}{1-C^2} \tag{5.9}$$

Since $0 \leq C < 1$, Equation 5.9 indicates that R_t eventually converges to a stable size of $\frac{T}{1-C^2}$.

5.4 Results

When compared with actual simulation results, the formulas above give a reasonable approximation of the growth of the reference lists over time (see Figure 5.1), as well as the growth of the corruption in the reference lists themselves (see Figure 5.2). The discrepancy between the predicted corruption and the simulated corruption arises in large part from our assumption that there are no collisions during the course of the protocol. In the actual simulation, if two peers nominate the same peer for the outer circle, it counts as only one nomination, not two. Since we give the adversary near-omniscience (i.e. the power to coordinate knowledge across all subverted peers), he can coordinate his outer-circle nominations to ensure that such collisions do not occur. The good peers do not have this luxury, and thus collisions occur disproportionately amongst the loyal outer-circle nominations, creating a net increase in the adversary's presence in the reference lists. Similarly, when churning in peers from the friends list, the current simulation code does not check for duplicates, so an attempt to churn in friends may select peers already present in the reference list, negating any benefit from this action. Indeed, simulation data shows that most collisions occur early on during the clustered simulations, since at that stage, each peer only knows about its friends, so that collisions happen frequently (see Figure 5.3). After enough time has elapsed, a peer has filled in its reference list with nominations from other peers, and so nominating peers from the reference list tends to cause fewer collisions.

To account for the collision effect, we model the nomination process as a balls-and-bins problem. Each bin represents a peer in the population, and we treat the

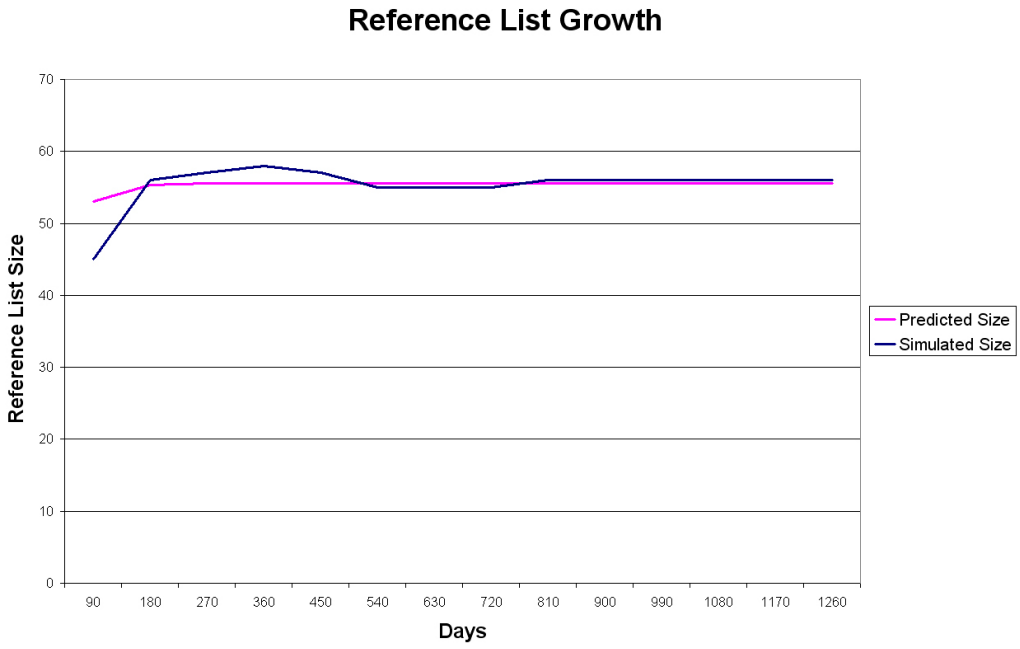

Figure 5.1: **Predicted vs. Simulated Reference List Growth** *Mathematical predictions of reference list growth dovetail closely with simulation data.*

nominations as balls thrown into the bins at random. Thus, we can adjust for the effect of colliding nominations by calculating the difference between the number of nominations (balls thrown), and additions to the reference list (the number of bins containing at least one ball) as follows:

Let m represent the number of bins and n represent the number of balls thrown. For a given bin, the probability of a ball landing in it is $\frac{1}{m}$, so the probability that none of the balls land in the bin is:

$$P_{empty} = (1 - \frac{1}{m})^n \approx e^{-\frac{n}{m}} \tag{5.10}$$

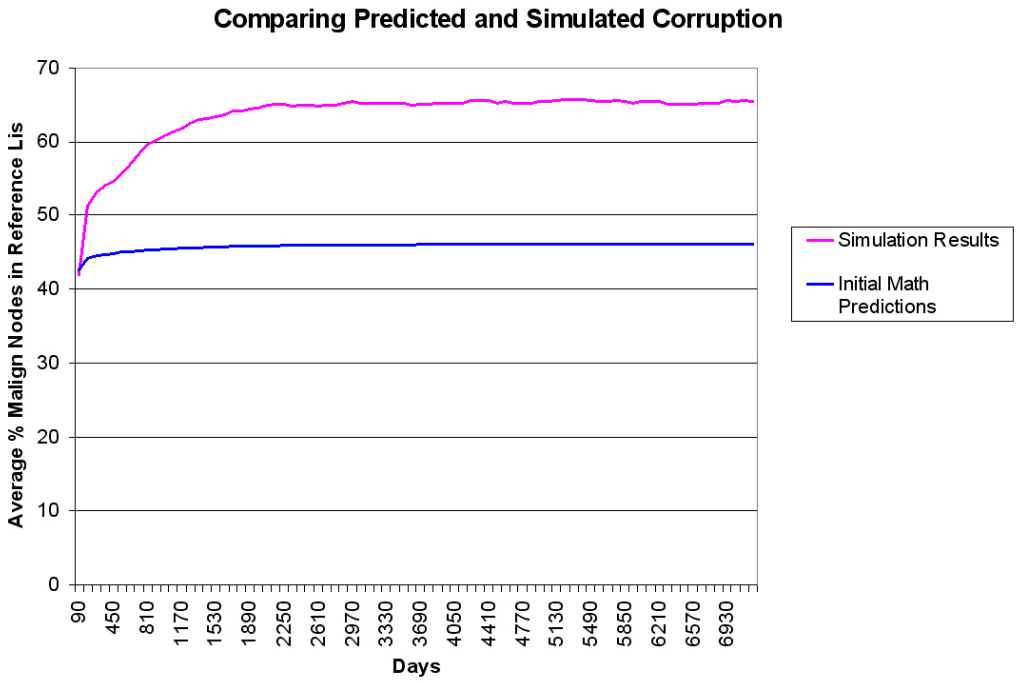

Figure 5.2: **Predicted vs. Simulated Reference List Corruption** *Simulation data detailing corruption of the reference lists, compared with predictions from the initial mathematical model.*

So for the general population, we have:

$$Expected\_Bins\_With\_Balls = (1 - P_{empty}) * m = (1 - e^{-\frac{n}{m}}) * m \tag{5.11}$$

Relating this back to our mathematical model, we find that we need to adjust our loyal nominations, Nom_{loyal}, such that:

$$Nom_{loyal} = (1 - e^{-\frac{Nom_{loyal}}{P}}) * P \tag{5.12}$$

However, we must also account for the fact that some of the nominated peers may already be in our reference list and/or our inner circle. The probability that a peer is in our reference list is $\frac{R}{P}$, and the probability that a peer is in our collection

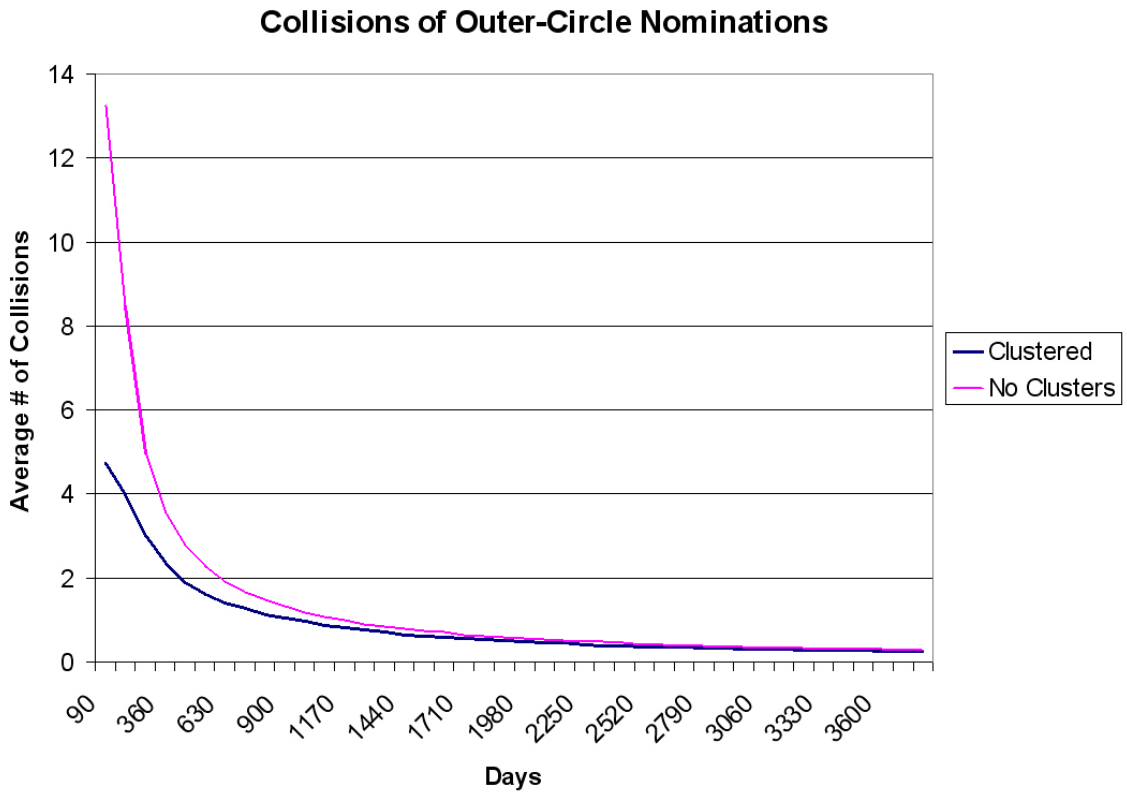

Figure 5.3: **Number of Collisions for Clustered vs. Unclustered Networks** *The number of collisions in the outer-circle nominations increases when the peers do not form clusters within the network.*

of nominations is $\frac{N}{P}$. For modeling purposes, we can consider these two probabilities independent, so the number of peers that are in our reference list and in the list of nominations, *overlap*, is

$$overlap = (\frac{R}{P} * \frac{N}{P}) * P = \frac{R * N}{P} \tag{5.13}$$

Subtracting *overlap* from Nom_{loyal} and performing a similar adjustment based on the size of the inner circle accounts for most of the collisions that take place.

As shown in Figure 5.4, adding this correction (and a similar correction for collisions during churning) improves the model's predictive powers, cutting the average error from 25% to under 10%. In this figure, we show two sets of simulation results. As discussed earlier, the LOCKSS system by default attempts to approximate the real structure of the Internet by clustering peers together (currently in groups of 30). The friends lists are then initialized such that the vast majority of entries in each peer's friends list come from within the peer's cluster, with the assumption that peers tend to befriend those "closer" in the network. Using this setup contradicts our analysis above, which assumed random choices distributed evenly across the population of peers. Since it is not entirely obvious that such a clustering provides the best model of network/social configurations, the simulation results in Figure 5.4 also show data from non-clustered simulation runs.

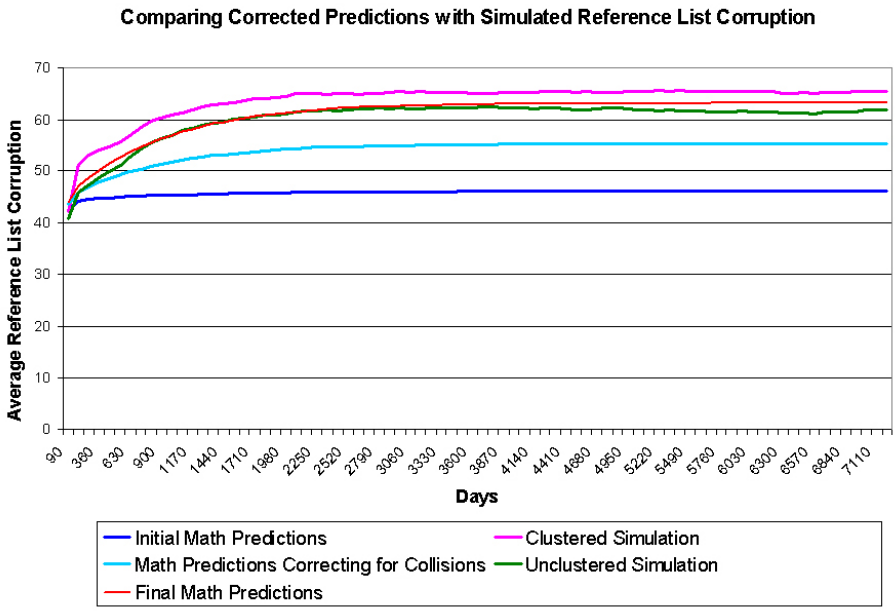

Figure 5.4: **Improved Predictions vs. Simulated Reference List Corruption** *Compares the simulation with the improved version of the mathematical model. This time, we correct for clustering and various types of collisions.*

As expected, eliminating the clusters and accounting for collisions significantly decreases the gap between predicted and simulated results. Interestingly, eliminating clusters actually increases the number of collisions in the early stages of the simulation (as shown in Figure 5.3). While this may seem counter-intuitive, the elimination of clusters provides a better random distribution over the population as a whole. Peers are more likely to have reference lists peers in common, due in large part to the effects of the birthday paradox. The clustering actually prevents global reach by partitioning the network into smaller sections, so after the initial set of collisions with friends, peers come into contact with peers from the opposite side of the network and the number of collisions drops rapidly.

As a final adjustment, we note that the LOCKSS protocol dictates that a peer involved in a poll (whether calling or participating) refuses any additional poll requests. Since the adversary will never need to ask for a replacement copy of the AU, he will never call a poll, and so he will never be too busy to respond to a poll request. However, if a loyal peer invites another loyal peer already occupied with a poll, then the poll initiator loses a potential loyal peer and the proportion of malign peers participating in the poll rises. As a rough estimate of the effect of "busy collisions", let us suppose that each poll consumes $pollTime$ hours of a peer's time and that each peer conducts a poll every Z months. As a further simplification, assume that every poll starts on a $pollTime$ boundary, i.e. polls start on discrete intervals rather than across a continuous spectrum. In that case, we only need to consider the probability that two polls start the same time. In a given year, one peer will call $\frac{12}{Z}$ polls, so altogether, the system will experience $\frac{12}{Z} * P$ polls over the course of the year. Each year has approximately $\frac{12 \cdot 30 \cdot 24}{pollTime}$ poll slots available, so assuming polls are randomly

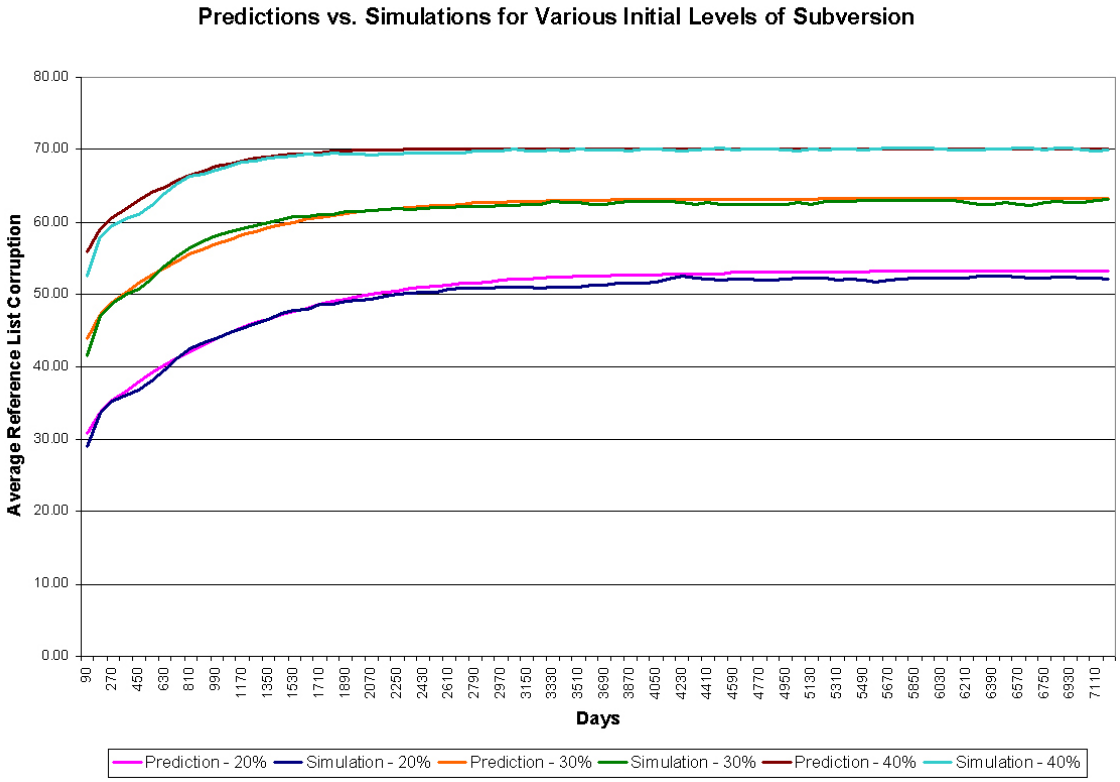

Figure 5.5: **Multiple Comparisons between Predicted Reference List Corruption and Simulated Reference List Corruption** *The predictions remain extremely accurate for varying levels of initial subversion.*

distributed across the slots, there will be:

$$pollsPerSlot = \frac{\frac{12}{Z} * P}{\frac{12 \cdot 30 \cdot 24}{pollTime}} = \frac{P * pollTime}{720Z} \tag{5.14}$$

In the current system, $pollTime \approx 5\ hours$ and $Z = 3\ months$, so $pollsPerSlot \approx 1.5$. Since the peer's poll counts towards occupying the poll slot, we expect a poll to collide with 0.5 other polls each round on average. Adding this adjustment to our calculations produces the line labeled "Final Math Predictions" in Figure 5.4. The figure clearly shows that this final adjustment brings our predictions into alignment with the simulation results, and indeed the numbers demonstrate an average error of less

than 1%. As shown in Figure 5.5, these predictions remain remarkably stable across varying initial conditions (less than 2% error in all three of the simulations). This suggests that the corrected mathematical model serves as an accurate and reliable predictor of long-term system behavior.

5.5 Implications

Developing a mathematical model in this fashion illustrates several important points. First, a sophisticated system such as LOCKSS involves complicated interactions that may have unexpected effects. For instance, the original LOCKSS design does not consider recommendation collisions, and yet we have seen that such collisions noticeably increase the corruption of the reference lists and make accurate modeling difficult. Some of these effects can be avoided. The collisions that occur during churning could be eliminated by churning in friends not already in the reference list. This would ensure that peers receive the full benefits of churning.

Next, the equations we developed suggest ways of improving the system's performance. We can simplify Equation 5.5 by reducing the number of variables involved, with the result that:

$$M_{oc} = X * \frac{M_{rt}}{R_t} * (2 - \frac{M_{rt}}{R_t}) \tag{5.15}$$

Substituting this into Equation 5.6 and simplifying, we have:

$$M_{r(t+1)} = -(\frac{X}{{R_t}^2}) * M_{rt}^2 + (1 - \frac{Q}{R_t} + \frac{2 * X}{R_t}) * M_{rt} + \frac{C * R_t * M_{r0}}{P} \tag{5.16}$$

This suggests several logical ways of improving the system's performance. Increasing the size of the population, P, will decrease the third term and hence will reduce the

growth of reference list corruption, since the adversary must subvert a larger absolute number of peers to have the same impact on the system. Similarly, increasing the number of votes necessary for a quorum, Q, will decrease the second term. Since $X = \frac{T}{1+C} - R_t + Q$, this will also increase the magnitude of the second-order term, further reducing the growth of reference list corruption. Intuitively, a larger quorum size means that more peers participate in each poll, and so the adversary must control even more peers in the system to infiltrate the reference lists. Finally, while we might hope that increasing the size of the reference list would improve our results, the equations indicate that the effect will be significantly more complicated and difficult to predict. It will depend in large part on the values of the other constants. Overall, these equations help highlight key design considerations not otherwise apparent and allow us to rapidly examine the long-term behavior of the system.

Chapter 6

Concluding Remarks

6.1 Future Work

Clearly, additional data on the frequency and severity of viruses, worms and other system compromises in the real world will benefit both the simulations and the analysis of the effects of a dynamic population on the system's performance. As researchers refine their monitoring techniques, we can incorporate their findings into our models and provide better predictions of LOCKSS' performance. 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 mathematical model of the adversary also requires further refinement and could be extended to other aspects of the adversary's behavior, or even to some of the other adversaries described in Section 2.2 - the nuisance adversary, the attritionist, etc. In general, developing a rigorous model of these adversaries may reveal additional vulnerabilities and suggest alternative remedies.

6.2 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 the 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.

Mathematical modeling provides an example of an underutilized approach to studying peer-to-peer adversaries. As our results studying LOCKSS illustrate, the rigorous analysis imposed by a mathematical model can lead to significant improvements in protocol design and system performance. A model can also provide a more intuitive understanding of the various factors that contribute to the system's overall behavior, revealing otherwise obscure vulnerabilities. Additionally, a mathematical model allows us to quickly examine long-term trends in the system's performance and make stricter security guarantees.

Finally, 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

This work would not have been possible without the guidance, experience and insights of my advisor, Professor Mema Roussopoulos. Discussions with the other members of the LOCKSS team - Charles Duan, Geoffrey Goodell, Rachel Greenstadt, and Ian Becker - also facilitated the development of my work. Professor Michael Mitzenmacher offered advice on the balls-and-bins portion of the mathematical model, and the EECS staff at Harvard provided assistance with hardware and software.

Finally, I would like to extend a special thanks to Diana Seymour for providing her assistance, editing skills, unwavering support and constant encouragement.

Bibliography

[1] Martín Abadi, Mike Burrows, Mark Manasse, and Ted Wobber. Moderately Hard, Memory-bound Functions. In *Proceedings of the 10th Annual Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, February 2003.

[2] William A. Arbaugh, William L. Fithen, and John McHugh. Windows of Vulnerability: A Case Study Analysis. *IEEE Computer*, 33:52–59, December 2003.

[3] The Internet Archive. The Internet Archive Wayback Machine. `http://www.archive.org/`.

[4] Association of Research Libraries. ARL Statistics 2000-01. `http://www.arl.org/stats/arlstat/01pub/intro.html`, 2001.

[5] Brooks Boliek. U.S. music industry sues song swappers. `http://www.reuters.co.uk/newsArticle.jhtml?type=internetNews&storyID=4643035&section=news`.

[6] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. In *OSDI: Symposium on Operating Systems Design and Implementation*. USENIX Association, Co-sponsored by IEEE TCOS and ACM SIGOPS, 1999.

[7] Miguel Castro and Barbara Liskov. Proactive Recovery in a Byzantine-Fault-Tolerant System. In *Fourth Symposium on Operating Systems Design and Implementation (OSDI)*, San Diego, CA, October 2000.

[8] John Douceur. The Sybil Attack. In *Proceedings of IEEE International Symposium on Peer-to-Peer Systems, IPTPS 02*, Cambridge, MA, March 2002. IEEE Computer Society.

[9] Cooperative Association for Internet Data Analysis. Telescope Analysis. `http://www.caida.org/analysis/security/telescope/`.

[10] The Long Now Foundation. The Rosetta Project. `http://www.rosettaproject.org/`.

[11] TJ Giuli and Mary Baker. Narses: A Scalable, Flow-Based Network Simulator. Technical Report arXiv:cs.PF/0211024, Computer Science Department, Stanford University, Stanford, CA, USA, November 2002.

[12] Andrew Goldberg and Peter N. Yianilos. Towards an Archival Intermemory. In *Proceedings of IEEE Advances in Digital Libraries, ADL 98*, pages 147–156, Santa Barbara, CA, 1998. IEEE Computer Society.

[13] Thomas 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.

[14] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. OceanStore: An Architecture for Global-scale Persistent Storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[15] Dahlia Malkhi and Michael Reiter. Byzantine Quorum Systems. *The Journal of Distributed Computing*, 11(4):203–213, October 1998.

[16] Petros Maniatis, Mema Roussopoulos, TJ Giuli, David S. H. Rosenthal, Mary Baker, and Yanto Muliadi. Preserving Peer Replicas By Rate-Limited Sampled Voting in LOCKSS. Technical Report arXiv:cs.CR/0303026, Stanford University, March 2003.

[17] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. Code-Red: A case study on the spread and victims of an Internet worm. In *Internet Measurement Workshop*, Marseille, France, September 2002.

[18] OpenBSD. OpenBSD Site. `http://www.openbsd.org/`.

[19] George Orwell. *1984*. New American Library, New York, New York, 1977.

[20] Honeynet Project. Know Your Enemy: GenII Honeynets. `http://project.honeynet.org/papers/gen2/`.

[21] LOCKSS Project. LOCKSS Home Page. `http://lockss.stanford.edu/`.

[22] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content Addressable Network. In *Proceedings of ACM SIGCOMM 2001*, San Diego, CA, August 2001.

[23] Eric Rescorla. Security Holes... Who cares? In *Proceedings of the 12th USENIX Security Symposium*, pages 75–90, Washington, DC, August 2003.

[24] David S. H. Rosenthal. LOCKSS Security. `http://lockss.stanford.edu/lockssecurity.html`.

[25] David S. H. Rosenthal, Petros Maniatis, Mema Roussopoulos, TJ Giuli, and Mary Baker. Notes on the Design of an Internet Adversary. In *Adaptive and Resilient Computing Security Workshop*, November 2003.

[26] Antony Rowstron and Peter Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proceedings of the 18th ACM SOSP'01*, Lake Louise, Alberta, Canada, October 2001.

[27] Emil Sit and Robert Morris. Security Considerations for Peer-to-Peer Distributed Hash Tables. In *International Workshop on Peer-to-Peer Systems*, March 2002.

[28] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 ACM SIGCOMM Conference*, pages 149–160, San Diego, CA, August 2001.

[29] Sun-Tzu. *The Art of War*. Oxford University Press, London, England, 1963.

[30] Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas. Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks. In *Proceedings of the 2003 ACM SIGCOMM Conference*, Karlsruhe, Germany, August 2003.

[31] The Memory Hole. Reasons Not to Invade Iraq. `http://www.thememoryhole.org/mil/bushsr-iraq.htm`.

[32] Dan S. Wallach. A Survey of Peer-to-Peer Security Issues. In *Proceedings of International Symposium on Software Security*, Tokyo, Japan, November 2002.

[33] Yang Wang, Deepayan Chakrabarti, Chenxi Wang, and Christos Faloutsos. Epidemic Spreading in Real Networks: An Eigenvalue Viewpoint. In *Proceedings of IEEE International Symposium on Reliable Distributed Systems, SRDS 03*, Florence, Italy, October 2003. IEEE Computer Society.