Predicting Adversary Infiltration in the LOCKSS System

Bryan Parno
and
Mema Roussopoulos

TR-28-04

Computer Science Group
Harvard University
Cambridge, Massachusetts

Predicting Adversary Infiltration in the LOCKSS P2P Digital Preservation System

Bryan Parno Mema Roussopoulos
Harvard University
{parno, mema}@eecs.harvard.edu

I. Motivation

In this paper, we develop an analytical model for the *lurking adversary* in the LOCKSS peer-to-peer digital preservation system [2] 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 protocol.

We develop a model of the lurking adversary that attempts to infiltrate the system by coloring each peer's view of the system. Specifically, the lurking adversary attempts to corrupt the *reference lists* of innocent peers. A peer's reference list is the list of peers that that peer knows and depends on to help it preserve the content it stores. The model predicts the growth of corruption in the reference list, allowing us to determine the probability that the adversary will cause permanent damage to the system.

We briefly describe the LOCKSS protocol. We then present our model and validate it using simulations and discuss ways to expand and improve this model.

II. 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.

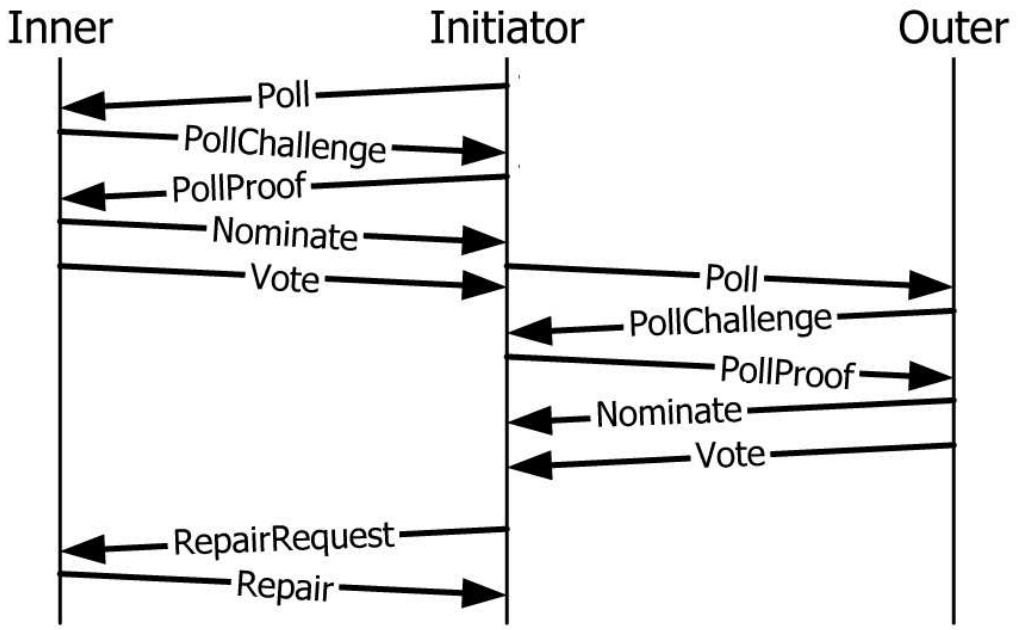

Fig. 1

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.*

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 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 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.

III. 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" [4]. 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 [5]. 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 **??**). 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.

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

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.

IV. 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 [3]. Later, we will explore the effects of altering this assumption.

V. Analysis

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

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

A. 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.

$$M_{r0} = \frac{M_0}{P} R_0 \tag{1}$$

$$M_F = \frac{M_0}{P} F \tag{2}$$

B. Calculation

Intermediate Steps

1) 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:

$$M_{ic} = \frac{M_{rt}}{R_t} N \tag{3}$$

$$L_{ic} = N - M_{ic} \tag{4}$$

2) 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:

$$M_{oc} = M_{ic} * \frac{X}{N} + L_{ic} * \frac{X}{N} * \frac{M_{rt}}{R_t} \tag{5}$$

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

$$M_{r(t+1)} = M_{rt} - \frac{M_{ic}}{N} * Q + M_{oc} + C * R_t * \frac{M_F}{F} \tag{6}$$

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:

$$R_{t+1} = R_t - Q + X + C * R_t \tag{7}$$

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 7 as:

$$R_{t+1} = C * R_t + \frac{T}{1+C} \tag{8}$$

We can eliminate the recurrence and express R_t as:

$$R_t = (F - \frac{T}{1-C^2}) * C^t + \frac{T}{1-C^2} \tag{9}$$

Since $0 \leq C < 1$, Equation 9 indicates that R_t eventually converges to a stable size of $\frac{T}{1-C^2}$.

VI. 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 2), as well as the growth of the corruption in the reference lists themselves (see Figure 3). 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 4). 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.

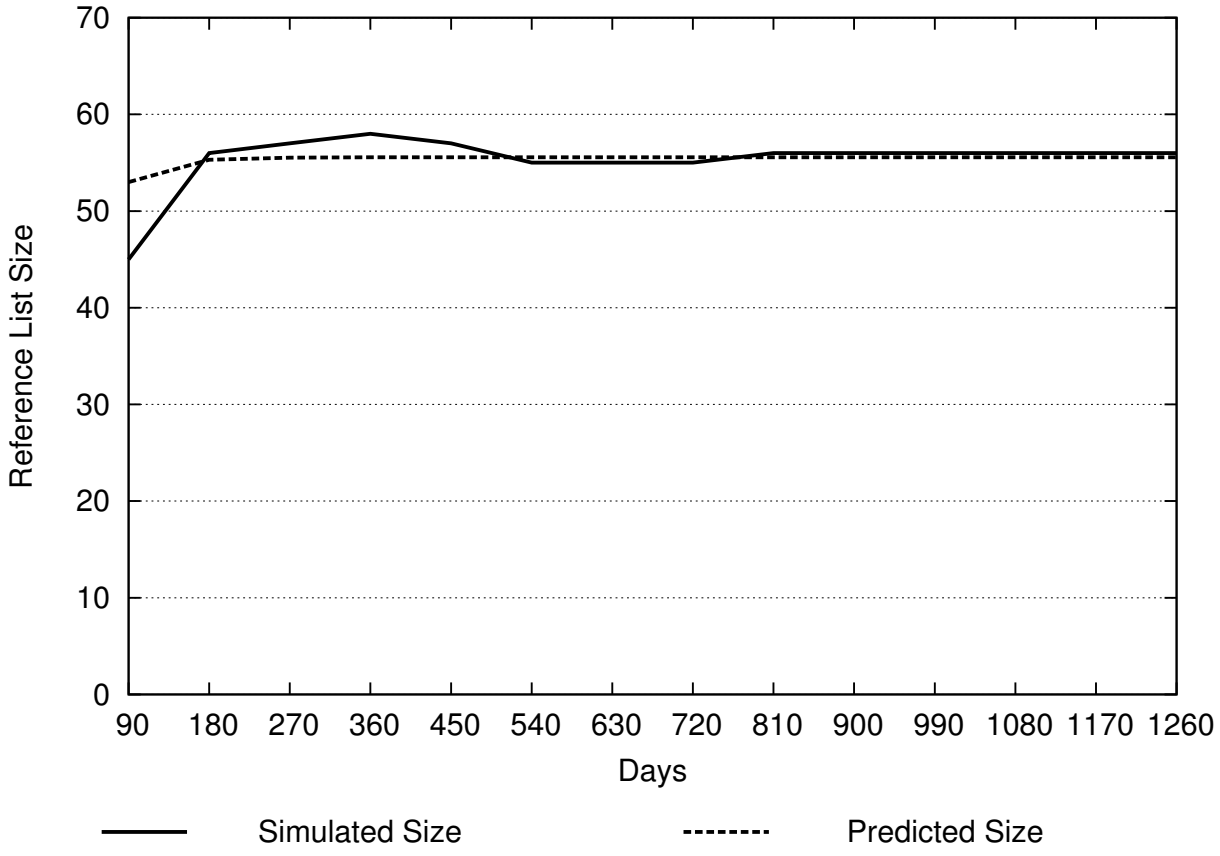

Fig. 2

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

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 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

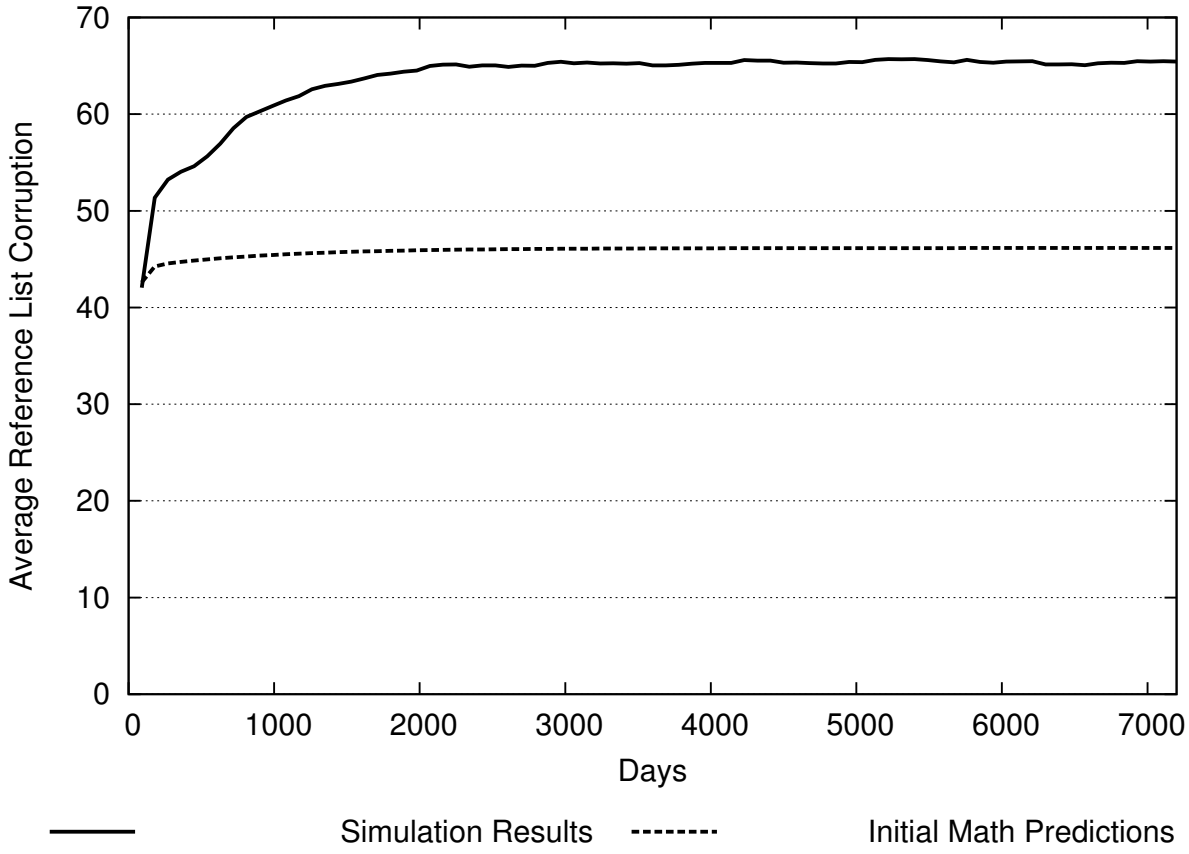

Fig. 3

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

probability of a ball landing in it is $\frac{1}{m}$, so the probability that none of the balls land in the bin is:

$$P_{empty} = (1 - \frac{1}{m})^n \approx e^{-\frac{n}{m}} \tag{10}$$

So for the general population, we have:

$$Expected\_Bins\_With\_Balls = (1 - P_{empty}) * m = (1 - e^{-\frac{n}{m}}) * m \tag{11}$$

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

$$Nom_{loyal} = (1 - e^{-\frac{Nom_{loyal}}{P}}) * P \tag{12}$$

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 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

$$overlap = (\frac{R}{P} * \frac{N}{P}) * P = \frac{R * N}{P} \tag{13}$$

Subtracting $overlap$ from Nom_{loyal} and performing a similar adjustment based on the size of the inner

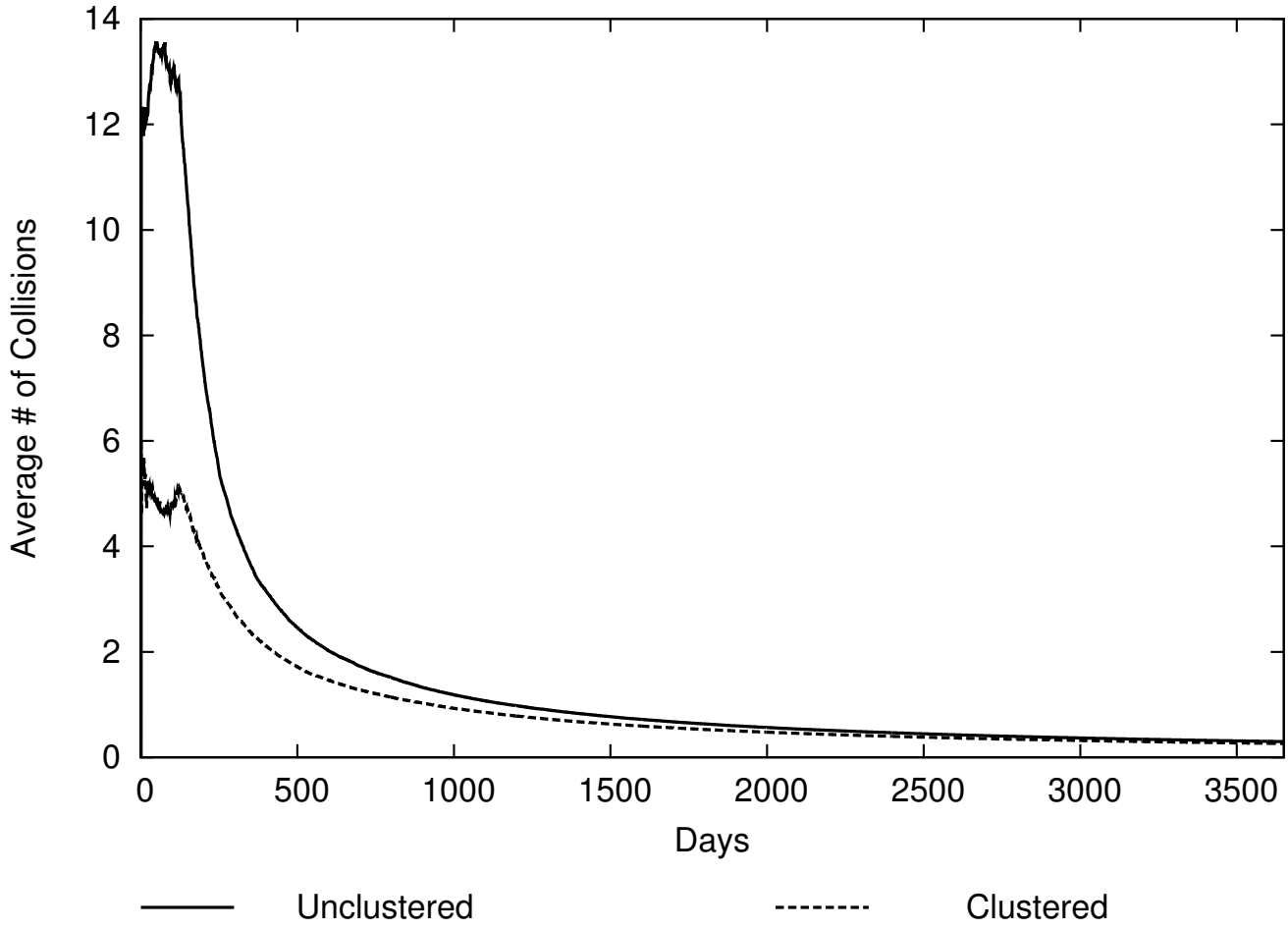

Fig. 4

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.*

circle accounts for most of the collisions that take place.

As shown in Figure 5, 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 also show data from non-clustered simulation runs.

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 4). 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 distributed across the slots, there will be:

$$pollsPerSlot = \frac{\frac{12}{Z} * P}{\frac{12 \cdot 30 \cdot 24}{pollTime}} = \frac{P * pollTime}{720Z} \tag{14}$$

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. The figure clearly shows that this final adjustment brings our predictions into alignment with

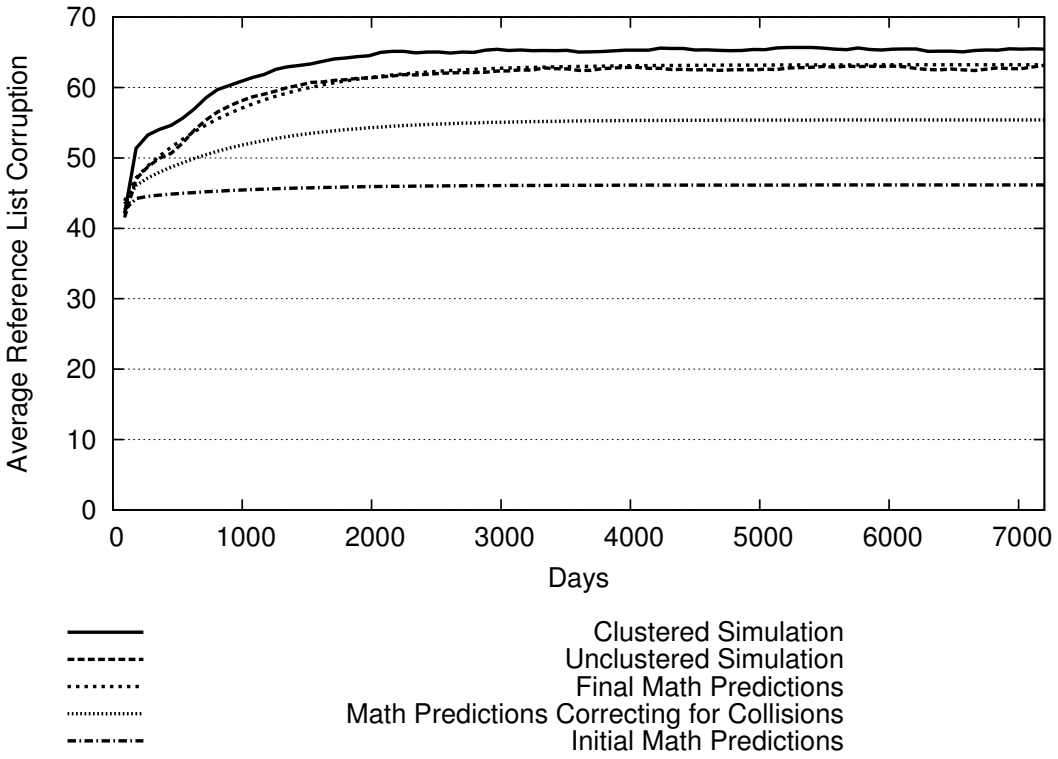

Fig. 5

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.*

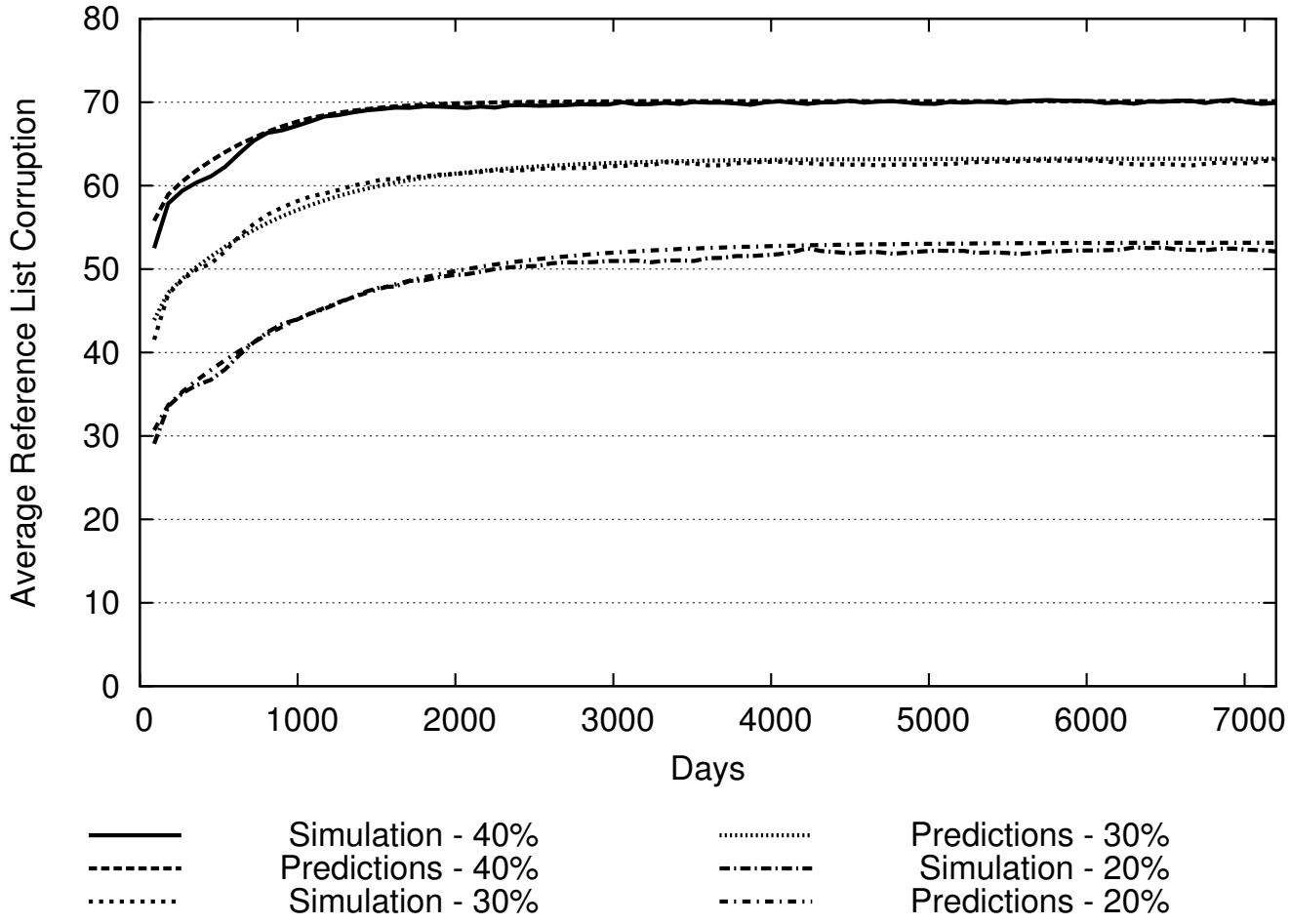

Fig. 6

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

the simulation results, and indeed the numbers demonstrate an average error of less than 1%. As shown in Figure 6, 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.

VII. 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 by reducing the number of variables involved, with the result that:

$$M_{oc} = X * \frac{M_{rt}}{R_t} * (2 - \frac{M_{rt}}{R_t}) \tag{15}$$

Substituting this into Equation 6 and simplifying, we have:

$$M_{r(t+1)} = -(\frac{X}{{R_t}^2}) * M_{rt}^2 + (1 - \frac{Q}{R_t} + \frac{2 * X}{R_t}) * M_{rt} + \frac{C * R_t * M_{r0}}{P} \tag{16}$$

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.

VIII. Acknowledgments

We would like to thank the other members of the Harvard LOCKSS team: Geoff Goodell, Rachel Greenstadt, Ian Becker, and Charles Duan. We also thank TJ Giuli for answering our questions about the LOCKSS code, Finally, we thank David Rosenthal, 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. Finally, we thank Diana Seymour for her assistance, editing skills, and support.

References

[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] Petros Maniatis, Mema Roussopoulos, TJ Giuli, David S. H. Rosenthal, Mary Baker, and Yanto Muliadi. Preserving Peer Replicas By Rate-Limited Sampled Voting. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 44–59, Bolton Landing, NY, USA, October 2003.

[3] 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.

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

[5] 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.