

Conservation vs. Consensus in Peer-to-Peer Preservation Systems

Prashanth P. Bungale, Geoffrey Goodell, and Mema Roussopoulos

Harvard University, Cambridge, MA 02138, USA
{prash, goodell, mema}@eecs.harvard.edu

Abstract. The problem of digital preservation is widely acknowledged, but the underlying assumptions implicit to the design of systems that address this problem have not been analyzed explicitly. We identify two basic approaches to address the problem of digital preservation using peer-to-peer systems: *conservation* and *consensus*. We highlight the design tradeoffs involved in using the two general approaches, and we provide a framework for analyzing the characteristics of peer-to-peer preservation systems in general. In addition, we propose a novel conservation-based protocol for achieving preservation and we analyze its effectiveness with respect to our framework.

1 Introduction

Recently, a number of peer-to-peer approaches have been proposed to address the problem of *preservation* (e.g., [4,7,11,2,3,6]). In their attempt to preserve some data so that it is available in the future, these systems face a number of challenges including dealing with natural degradation in storage media, catastrophic events or human errors, attacks by adversaries attempting to change the data preserved, as well as providing incentives to other peers to help in the preservation task. These systems differ in their approaches and the systems' designers characterize their approaches in different ways: archiving, backup, digital preservation. But these peer-to-peer systems share a basic premise: that each peer is interested in preserving one or more *archival units* (AUs) and uses the aid and resources of other peers to achieve its goal.

In this paper we provide a framework for analyzing the characteristics and highlighting the design tradeoffs of peer-to-peer preservation approaches. Suppose that our preservation system involves each AU of interest being replicated on a subset of the peer population. Consider a particular archival unit being replicated on a subset consisting of n peers, denoted (p_1, p_2, \dots, p_n) . We use $p_i(t)$ to denote the copy of the archival unit held by peer p_i at time t . To simplify the scenario somewhat, presume that all peers enter the system at time t_0 . We assert that there are two basic approaches to providing preservation:

- **CONSENSUS.** The goal is for all peers in the system to come to a uniform agreement over time; that is to say that as $t \rightarrow \infty$, we have that $\forall i, j : p_i(t) = p_j(t)$. In essence, each peer always believes that the version of the

AU it has may be questionable and is willing to use the aggregate opinion of the community to influence its own copy, even if that sometimes involves replacing the current copy with a new one.

- CONSERVATION. The goal is for each peer to retain indefinitely the exact copy of the AU that it holds initially; that is to say that as $t \rightarrow \infty$, we have that $\forall i, t : p_i(t) = p_i(t_0)$. In essence, each peer believes that the version of the AU it starts with is the “right” version, and it always attempts to preserve this copy, even if other peers disagree. When it suffers a damage to its AU, it seeks the help of other peers to recover this right version.

There is a fundamental trade-off between these two approaches. If a peer happens to have a wrong version, conserving the data as it is is detrimental to preservation, whereas consensus helps preserve the right version if the other peers happen to supply the right version as the consensus version. On the other hand, if a peer happens to have the right version, conserving the data as it is helps preserve the right version, whereas consensus can potentially cause it to get infected with a wrong version (if the other peers happen to supply a wrong version to it as the consensus version).

2 Framework for Design Considerations

The design choice between conservation and consensus is not straightforward, but involves balancing and prioritizing various conflicting goals and choosing the best suited approach. To aid this process, we discuss below a list of considerations for designing a peer-to-peer preservation system. There may be other useful considerations, but we’ve found this list to be particularly useful.

Trust in the source of the AU. If the original source of the AU is perfectly trusted to supply the right version of the AU always, consistently, to all the subscriber peers (i.e., peers that will hold replicas of this AU), conservation might be a better preservation strategy. On the other hand, if the source supplies the right version to some subscriber peers and a wrong version to some others, consensus could help, as long as the subscribers with the right version outnumber those with a wrong version and are thus able to convince those with the wrong version to replace their archived documents.

Trust in the means of procuring the AU. If peers in the system use an unreliable means of obtaining the AUs to be archived, then it is likely that only a fraction of the peers will obtain the correct copy at the outset. This circumstance may provide an argument in favor of a consensus-based approach, since conservation alone will lead to preservation of invalid copies.

Frequency of storage faults. If storage degradation is frequent because of the environment or particular storage medium chosen, then, it could prove difficult to achieve consensus on an AU. This is because if a substantial portion of peers are in damaged state at any point of time, then a deadlock situation could arise. The peers need to get a consensus copy to recover from their damage, and on the other

hand, the peers need to first recover from their damage in order to achieve good consensus. Thus, the consensus approach may not be well-suited for systems with high frequencies of storage faults. On the other hand, a conservation approach might avoid this problem because all it requires to recover from a damage is any one peer being able to respond with the AU being conserved.

Frequency of human error. If system operators are likely to commit errors, for instance, while loading an AU to be preserved or while manually recovering the AU from a damage occurrence, conservation could be detrimental because the system may end up preserving an incorrect AU, whereas consensus could help recover the right AU from other peers.

Resource relevance to participants. Relevance [10] is the likelihood that a “unit of service” within a problem (in our case, an archival unit) is interesting to many participants. When resource relevance is high, both consensus and conservation could benefit from the relevance and would be equally suitable. However, when the resource relevance is low, because cooperation would require artificial or extrinsic incentives to make the peer-to-peer solution viable, conservation would be better suited as it would require less frequent interactions (specifically, only during recovery from damage) and smaller number of peers participating as compared to consensus.

Presence of adversaries. Preservation systems may be subject to various attacks from adversaries. We focus on two kinds of attacks that exploit peer interactions in the system: *stealth-modification attack* and *nuisance attack*. In a stealth-modification attack, the adversary’s goal is to modify the data being preserved by a victim peer, but without being detected. In a nuisance attack, the adversary’s goal is to create nuisance for a victim peer, for instance by raising intrusion detection alarms that may require human operator intervention. The design of a preservation system that takes these attacks into account would involve the following two considerations:

- ***Tolerance for stealth-modification:*** Is it acceptable to the users of the preservation system for some peers being successfully attacked by a stealth modification adversary, and possibly recovering eventually? i.e., Is it tolerable for some of the peers to have an incorrect AU sometimes? If the answer is ‘yes’, then both conservation and consensus may be equally suitable approaches. But, if the system has very low tolerance for stealth-modification attacks, conservation may be appropriate as it is less influenced by (and thus, less susceptible to) other peers. Consider the case in which there is substantial likelihood that adversaries may have subverted peers, or if there is fear that adversarial peers form a large percentage of the overall peer population. In this circumstance, consensus is a dangerous strategy because it may cause all of the well-behaved peers that have the right version to receive an invalid version, and thus conservation may be appropriate. However, there is also a downside to using conservation in that once the adversary is somehow able to carry out a stealth-modification attack successfully, the victim peer, by definition, believes that

its copy is the right one and is thus prevented from being able to recover, even after the adversary has stopped actively attacking it.

- ***Tolerance for nuisances***: Can the users tolerate frequent nuisances? The frequency of possible nuisance attacks is limited by the frequency of invoking peer participation. Thus, if there is low tolerance to nuisance attacks, then a conservation approach may be preferable because each peer relies on other peers only when it suffers a damage.

3 LOCKSS - An Example of the Consensus Approach

In this section, we consider LOCKSS, an example of a preservation system following the consensus approach, and discuss its design with respect to our framework.

The LOCKSS system [7,9] preserves online academic journals using a peer-to-peer auditing mechanism. The system provides a preservation tool for libraries, whose budgets for preservation are typically quite small [1]. Each (library) peer crawls the websites of publishers who have agreed to have their content preserved and downloads copies of published material (e.g. academic journals) to which the library in question has subscribed. The cached information is then used to satisfy requests from the library's users when the publisher's website is unavailable.

Web crawling is an unreliable process, making it difficult for peers to determine without manual inspection of the crawled material whether complete and correct replicas of the AUs of interest have been downloaded. Peers therefore need some automated way to determine if their copy is correct. LOCKSS uses consensus for this purpose. Peers perform sampled-auditing of their local copies to ensure that it agrees with the consensus of peers.

The LOCKSS design is based on the following characteristics and/or assumptions in our design framework:

Trust in the source of the AU and trust in the means of procuring the AU: low, as long as only a relatively small portion of the overall peer population initially acquires an incorrect AU either from the source or through the procurement means.

Frequency of storage faults: extremely low (assumed to be once in 200 machine years on an average); *Frequency of human error*: can be high; *Resource relevance to participants*: high (as libraries often subscribe to the same AU's from the publishers).

Presence of adversaries: at most one-third to 40% of the peer population could be adversarial; the adversary is assumed to have unlimited computation power and unlimited identities. Tolerance for stealth-modification and for nuisances: *medium*.

Looking at these characteristics and assumptions, and considering the suitability of the approaches described in our design framework, we can clearly see why the system designers have chosen the consensus approach. We describe below the design of the consensus protocol of LOCKSS, and discuss the factors relevant to our framework on the way.

Each peer maintains two lists: a *friends list* and a *reference list*. The reference list is a list of peers that the peer in question has recently discovered in the process of participating in the LOCKSS system. The friends list is a list of peers (*friends*) that the peer knows externally and with whom it has an out-of-band relationship before entering the system. When a peer joins the system, his reference lists starts out containing the peers on his friends list.

Periodically, at a rate faster than the rate of natural bit degradation, a peer (the *poller*) conducts an *opinion poll* on an AU. The peer takes a random sample of peers as a *quorum* from its reference list and invites the chosen peers as *voters* into a poll. The voters vote on the AU by sending hashes of their individual copies of the AU to the peer initiating the poll. The poller compares the votes it receives with its local copy. If an overwhelming majority of the hashes received agrees with the poller's hash, then the poller concludes that its copy is good, (i.e., it agrees with the consensus) and it resets a refresh timer to determine the next time to check this AU. If an overwhelming majority of hashes disagree, then the peer fetches a *repair* by obtaining a copy of the AU from one of the disagreeing peers and re-evaluating the votes it received. That is, the peer alters its copy of the AU so that it agrees with the consensus. If there is neither landslide agreement nor landslide disagreement, then the poll is deemed *inconclusive* and the poller raises an alarm.

Because natural storage degradation is assumed to be a relatively infrequent occurrence, it is unlikely that many peers will simultaneously be experiencing degradation. If an inconclusive poll results, it is an indication that an attack might be in progress. LOCKSS uses alarms as a way of performing intrusion detection, so that when an attack is suspected, humans are called upon to examine, heal, and restart the system. This requirement of humans being expected to examine, heal, and restart the system every time an alarm is raised, which could happen on every poll in the theoretically worst case, is the reason why the system cannot tolerate frequent nuisance attacks. Therefore, the designers aim for nuisance attacks being only infrequently possible.

At the conclusion of a poll, the poller updates its reference list as follows. First, it removes those peers that voted in the poll so that the next poll is based on a different sample of peers. Second, the poller replenishes its reference list by adding *nominated peers* and peers from the friends list. Nominated peers, or *nominees*, are peers that are introduced by the voters when the voters are first invited to participate in the poll. Nominees are used solely for discovery purposes so that the poller can replenish its reference list. Nominees vote on the AU, but their votes are not considered in determining the outcome of the poll. Instead, their votes are used to implement admission control into the reference list. Nominees whose votes agree with the poll outcome are added to the reference list.

The bias of friends to nominees added is called *churn*. The contents of the reference list determine the outcome of future polls. Adding more friends to the reference list than nominees makes the poller vulnerable to targeted attacks aimed at its friends. Adding more nominees than friends to the reference list increases the potential for Sybil attacks [5].

Using a combination of defense techniques such as rate-limitation, effort-balancing, reference list refreshes and churn, among others, the LOCKSS protocol achieves strong, but imperfect, defense against a stealth-modification adversary. Experimental results show that the probability that, at any point in time, the user at a peer would access a bad AU was increased by just 3.5%. However, it was also observed that around one-third of the loyal (i.e., non-adversarial) peers end up being attacked by a stealth-modification adversary who starts with an initial subversion of 40% of the overall peer population. Although the LOCKSS authors have reported that successful nuisance attacks have been observed to be seldom, they have not looked into what exactly happens when an alarm is raised at a peer (i.e., to what extent the adversary is rooted out), and so we cannot analyze the real impact of nuisance attacks at this time.

4 Sierra - An Example of the Conservation Approach

The key notion of the conservation approach is that each peer, being fully confident that the version of the AU it stores is the right version, attempts to conserve its own version. To do so, the peer ignores what the version may look like at other peers, except when it suffers a “bit-rot”, i.e., a storage failure or some other event that results in its AU being damaged, at which point it looks to other peers for recovery.

Given just the conservation notion, one might consider a simple solution for implementing conservation such as storing the AU, along with a signed hash of the AU remotely on other peers, and relying on this information while recovering from a bit-rot. This solution may be perfectly acceptable in peer-to-peer backup applications. However, in a LOCKSS-like application that would want to exploit the high resource relevance existing in the system (to reduce unnecessary storage overhead) and avoid long-term secrets (which may be unreasonable for long-term preservation on the order of decades), this simple solution may not be suitable.

We propose Sierra as a conservation-based alternative to the LOCKSS protocol. Sierra shares some features with LOCKSS in that it exploits resource relevance and does not depend on long-term secrets. It also borrows some techniques from LOCKSS such as calling opinion polls using a sample of the peer population. However, Sierra’s primary goal departs fundamentally from that of LOCKSS. While Sierra makes use of opinion polls (which have a consensus flavor), it does not blindly rely on the results of the polls. We thus refer to Sierra as using a *tamed-consensus* approach towards achieving the conservation goal.

Following are the characteristics and/or assumptions we use that are relevant to our design framework:

Trust in the source of the AU and trust in the means of procuring the AU: high; *Frequency of storage faults:* low; *Frequency of human error:* low; *Resource relevance to participants:* high.

Presence of adversaries: up to 60% of the peer population could be adversarial; the adversary is assumed to have unlimited computation power and unlimited identities; Tolerance for stealth-modification: *zero-tolerance*; Tolerance for nuisances: *low*.

Since we prioritize allowing higher presence of adversaries, and yet having zero-tolerance for stealth-modification attacks and low tolerance for nuisance attacks, we are forced to make the stronger assumption of high trust in the source and procurement means for the AU.

Since a conservation-based system assumes complete confidence in the local AU, a bit-rot occurrence is the only “*time-of-need*” when a peer might have to rely on the other peers to recover its AU. During the remaining time, the peer would be “*self-sufficient*” in terms of preserving the AU. Alongside each stored AU, a peer stores a hash of that AU and periodically checks the AU against the hash to determine if it is self-sufficient or in its time of need.

In addition, we introduce a host of defense techniques to help a peer *conserve* its AU. Peers call polls periodically as in LOCKSS. If the stored AU and hash match, then the poller ignores the result of the poll. However, the poller updates its reference list as in the LOCKSS protocol with the following change. Any voters whose votes disagree with the poller’s AU are removed from the reference list and also *blacklisted* from providing votes to this poller in the future.

If the AU and local hash do not match when the poller calls its next poll, it enters a “time-of-need” state and remains in this state for the next n polls, where n is a system-defined parameter. During (and only during) a time-of-need poll, the poller checks to see if any of the peers that are in the minority agree with each other. If a *minority threshold* number of peers agree with each other, the poller raises an alarm to notify its local operators. Otherwise, the poller repairs using the version of the AU stored by the majority. A minority alarm indicates that either the majority or the minority is potentially adversarial. When this alarm is raised, the operator is expected to examine and choose the right one among the different contending versions of the AU and then, the peers who supplied the incorrect versions will be blacklisted. Note that the larger n is, the more likely a stealth-modification attack will be detected because the higher the chance that the poller will find, in a subsequent poll, a minority threshold number of peers that agree with each other.

In Sierra, voters only vote if they are in the self-sufficient state (i.e., their stored AU and hash match) and decline the poll invitation otherwise.

4.1 Analysis

The Sierra protocol uses the basic underlying features of the LOCKSS protocol for calling polls and managing the peer-to-peer network, and thus to analyze its effects theoretically, we start by examining existing theoretical properties of LOCKSS. Due to lack of space, we omit the details of the LOCKSS analysis [8] here.

Attaining a presence in a victim peer’s reference list is the only means through the protocol by which an adversary can launch a stealth-modification or a nui-

symbol	default description
C	0.1 churn rate (ratio)
M_0	1000 initial number of malign peers
Q	100 quorum # of voters needed per poll
P	10000 total population
T	600 reference list size
Int	3 months mean inter-poll interval

Fig. 1. Parameters for Sierra analysis

sance attack. We call the strength of adversarial presence, i.e., the proportion of reference list peers that are adversarial, the adversary's *foothold*. The only way for an adversary to attain higher foothold in a peer's reference list is to *lurk*, i.e., to *act* loyal (or non-malign) by voting using the correct version of the AU and nominating its minions for entrance into the poller's reference list.

Consider an adversary in LOCKSS that lurks. We can model the expected number of malign (i.e., adversarial) peers, M_{rt} , in a loyal peer's reference list at time t , given a uniform distribution of adversaries throughout the population, as a function of time and a set of system parameters (See Figure 1) [8]:

$$M_{r(t+1)} = -\frac{X}{T^2}M_{rt}^2 + \left(1 - \frac{Q + 2X}{T}\right)M_{rt} + \frac{CTM_0}{P} \quad (1)$$

where X , the expected number of nominees in steady-state equilibrium, is given by:

$$X = Q + T \left(\frac{1 - C^2}{1 + C} - 1 \right) \quad (2)$$

However, because Sierra introduces blacklisting as a means by which a peer may eradicate those who vote with invalid copies from its reference list, the recurrence equation for Sierra is somewhat different. The only opportunity for an adversary to have its set of malign peers (p_{m1}, \dots, p_{mk}) vote with an invalid copy and still increase its expected foothold in the reference lists of some target peer p_t occurs when p_t suffers a bit-rot and enters its time-of-need state.

Suppose that μ is the threshold for raising an alarm in the event of minority agreement. Given that a stealth-modification adversary seeks to win a poll and avoid detection, the malign peers must vote with the invalid copy of the AU only if there exist at least $Q - \mu$ malign peers in a given poll called by p_t , and further if the poll happens to be a time-of-need poll. Otherwise, if the adversary attacks with its bad copy, it ends up losing all of its hard-earned foothold due to blacklisting. Therefore, the optimal strategy for the stealth-modification adversary in the case where there are less than $Q - \mu$ malign peers in a poll is to lurk, so that it can try to increase its foothold further. Thus, the recurrence equation does not change for the stealth-modification adversary if we assume an optimal adversary strategy (and therefore no blacklisting).

If an adversary wants to simply create a nuisance that raises an alarm, then at least μ malign peers must vote with a non-majority copy of the AU. Since the act of creating a nuisance does not benefit from having more than μ peers vote with the invalid copy, it is in the best interest of the adversary to have only μ peers, $(p_{m1}, \dots, p_{m\mu})$ perform this task. The adversary would now lose some foothold due to blacklisting whenever it creates a nuisance, and therefore, the recurrence equation changes. Next, we introduce three other variables: F , the mean time between failures (in terms of number of polls) for the particular storage system, d , the likelihood that an adversary will choose to create a nuisance when it can, and K , the likelihood that an adversary will create a nuisance even when a peer is *not* in time-of-need. K represents the extent to which the adversary has knowledge of when a bit-rot occurs for a particular loyal peer. If the adversary has perfect knowledge (through some covert channel or out-of-band means), then $K = 0$, but we believe that in most realistic cases, K would be closer to 1. Whenever the adversary tries to create a nuisance for a given peer p_t by supplying μ malicious votes, p_t will evict μ adversarial peers from its reference list. Thus, our new recurrence is represented by the following equations:

$$M'_{r(t+1)} = -\frac{X}{T^2}M_{rt}^2 + \left(1 - \frac{Q + 2X}{T}\right)M_{rt} + \frac{CTM_0}{P} \quad (3)$$

$$M_{rt} = M'_{rt} - \frac{d\mu(1 + K(F - 1))}{F} \quad (4)$$

Since we are interested in powerful adversaries, we assume for the rest of our analysis that an adversary somehow has perfect knowledge about peers suffering bit-rots and will attack or create a nuisance only when a peer is in time-of-need.

Effectiveness against Stealth Modification Attacks. We first consider the question of what conditions can actually lead to an adversary being able to carry out a stealth-modification attack successfully (i.e., without being detected). An attack is possible only if:

- The adversary has somehow achieved very high (close to 100%) foothold in the victim's reference list – because it would otherwise be detected through the minority threshold alarm within the n polls called during the time-of-need.
- More importantly, the adversary is able to sustain that foothold for a sufficient number of consecutive polls – specifically, during the n time-of-need polls.
- The adversary is able to somehow magically attack exactly when a damage has just occurred at the victim, i.e., should have perfect knowledge of the victim's damage occurrences.

We now use the mathematical model discussed earlier to show that the adversary is not able to carry out stealth-modification attacks successfully. Recall that the the optimal adversary strategy for stealth-modification is lurking continuously until it attains enough foothold. We find that the adversary is unable

to lurk and attain footholds high enough (i.e., enough to ensure at least $Q - \mu$ poll invitations) to be able to carry out successful attacks. Figure 2 shows the result of using equations 3 and 4 with $d = 0$ to obtain the equilibrium foothold value (which is the maximum expected number of malicious peers on the reference list of a given loyal peer) for different initial subversion values. As we can see from this graph, even for initial subversions as high as 60%, the equilibrium foothold never reaches 80%, which is the foothold required to ensure at least $Q - \mu$ poll invitations.

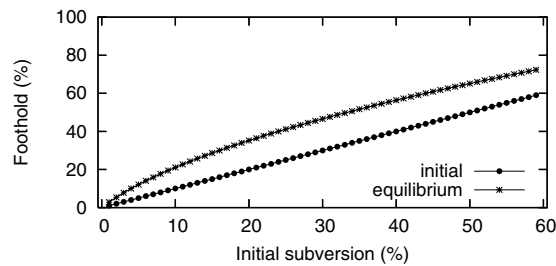


Fig. 2. EQUILIBRIUM FOOTHOLD ACHIEVED WITH VARYING INITIAL SUBVERSION. *Pre-suming MTBF $F = 10$ years, nuisance probability $d = 0$, and minority threshold $\mu = 20$.*

Note that we have assumed that the adversary has perfect knowledge of the victim's damage occurrences. If the adversary has no out-of-band means to acquire this knowledge, it is close to impossible for the adversary to be able to lurk for the entire period that the victim peer is healthy (to avoid being blacklisted) and then attack exactly when it suffers a damage.

Effectiveness against Nuisance Attacks. First, we note that in Sierra, the maximum frequency at which an adversary can create nuisance is limited to once every bit-rot occurrence instead of once every poll as in LOCKSS. Next, we observe that creating a nuisance comes with an associated penalty: peers voting with an invalid copy are blacklisted by the operator upon being notified by the alarm, and they cannot return to the reference list. We want to show that the penalty associated by blacklisting creates some disincentive for nuisance attacks. For the following analysis, we consider adversaries having an initial subversion of 10% of the peer population. First, this subversion ratio is enough for the adversary to be able to carry out nuisance attacks. Second, while an adversary with a higher subversion ratio could very well carry out nuisance attacks, it does not lose much foothold because it can quickly make up for the loss it suffers (due to blacklisting) by nominating its minions.

Figure 3 shows what happens when we vary the probability in which an adversary creates a nuisance. Observe that even if an adversary has complete

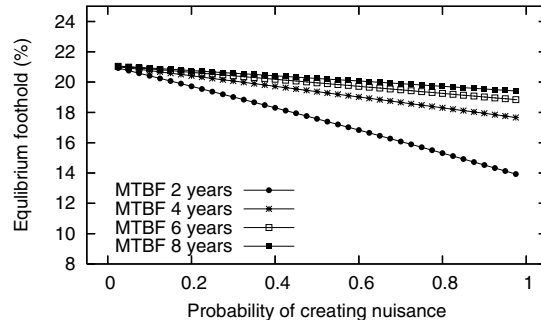


Fig. 3. VARYING NUISANCE PROBABILITY. Presuming 10% initial subversion and minority threshold $\mu = 20$.

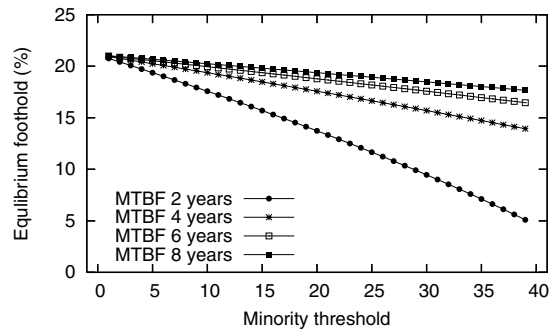


Fig. 4. VARYING MINORITY THRESHOLD. Presuming 10% initial subversion and nuisance probability $d = 1$. Note that the x -axis shows the minority threshold as an absolute number of peers and the quorum is 100.

knowledge of when a given peer's AU is damaged, it may still have substantial incentive not to create a nuisance too frequently, particularly if the MTBF is sufficiently short that the adversary does not have time to restore its representation on the peer's reference list between successive failures.

Finally, Figure 4 shows the effect of varying the minority threshold. We see that with lower minority thresholds, the adversary incurs lesser penalty and therefore, the adversary has less of an incentive not to create a nuisance. On the other hand, we know intuitively that increasing the threshold, while contributing to a better defense against nuisance attacks, leads to more opportunities for stealth modification attacks.

5 Conclusions

Preservation is not a straightforward problem. Peer-to-peer systems aimed at providing a preservation solution face a number of conflicting design considerations

that force designers to make difficult choices. We have presented a framework for considering the tradeoffs involved in designing such a system. We have also discussed two example systems with respect to this framework, LOCKSS and Sierra, that embody the two basic approaches to preservation: consensus and conservation, respectively. We observe that LOCKSS allows assumptions of distrustful source and procurement means for the AU, while achieving moderately strong defense against stealth-modification and nuisance attacks. On the other hand, Sierra achieves much stronger defense against both attacks, but at the expense of making assumptions of high trust in the source and procurement means for the AU.

Acknowledgments

We would like to thank the following people for their very helpful feedback and suggestions: Mary Baker, T. J. Giuli, Rachel Greenstadt, Petros Maniatis, Radhika Nagpal, Bryan Parno, Vicky Reich, and David S. H. Rosenthal.

References

1. ARL – Association of Research Libraries. ARL Statistics 2000-01. <http://www.arl.org/stats/arlstat/01pub/intro.html>, 2001.
2. T. Burkard. Herodotus: A Peer-to-Peer Web Archival System, Master's thesis, MIT, Jun 2002.
3. B. F. Cooper and H. Garcia-Molina. Peer-to-peer data preservation through storage auctions. *IEEE Transactions on Parallel and Distributed Systems*, to appear.
4. Landon P. Cox and Brian D. Noble. Samsara: Honor Among Thieves in Peer-to-Peer Storage. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pages 120–132, Bolton Landing, NY, USA, October 2003.
5. J. Douceur. The Sybil Attack. In *1st Intl. Workshop on Peer-to-Peer Systems*, 2002.
6. HiveCache, Inc. Distributed disk-based backups. Available at <http://www.hivecache.com/>.
7. P. Maniatis, M. Roussopoulos, TJ Giuli, D. S. H. Rosenthal, M. Baker, and Y. Muliadi. Preserving Peer Replicas By Rate-Limited Sampled Voting. In *SOSP*, 2003.
8. B. Parno and M. Roussopoulos. Predicting Adversary Infiltration in the LOCKSS System. Technical Report TR-28-04, Harvard University, October 2004.
9. D. S. H. Rosenthal, M. Roussopoulos, P. Maniatis, and M. Baker. Economic Measures to Resist Attacks on a Peer-to-Peer Network. In *Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, June 2003.
10. M. Roussopoulos, TJ Giuli, M. Baker, P. Maniatis, D. S. H. Rosenthal, and J. Mogul. 2 P2P or Not 2 P2P? In *IPTPS*, 2004.
11. D. Wallach. A Survey of Peer-to-Peer Security Issues. In *Intl. Symp. on Software Security*, 2002.