Preserving Peer Replicas By Rate-Limited Sampled Voting[6]

Peer to Peer Seminar

Renata Dividino

Email: dividino@stud-cs.saarland.de

Table of contents

Introduction

Publishers migrate to the Web allowing readers to access digital documents. Unfortunately cannot guarantee a long-term online access to those digital documents. Libraries though consider it one of their responsibilities. Libraries subscribe different publisher, acquire from them lots of copies of documents and distribute them to local readers providing long-term online access.

In order to guarantee long-term access of non-corrupt document's copies to local reader, libraries shall cooperate with each order to preserve their content. Having that more than one library hold the same material, they can, by working together, detect and repair damage on their content.

Digital Preservation Systems is a long-term large-scale application that models the physical document system into a Web-published document system providing tools for libraries (web-caches) to collect the material to which they subscribe, distribute it to local readers and preserve it by cooperating with others libraries. The goal of a digital preservation system is to prevent change.

The Digital Preservation system LOCKSS (Lots of Copies Keep Stuff Safe) allows a large number of independent, low-cost, persistent web caches that cooperate to detect and repair damage by voting in "opinion polls" on their cached documents. The voting protocol using by LOCKSS is a peer-to-peer opinion poll protocol that addresses scaling and attack resistance issues.

Design Principles

Digital preservation systems have some unusual features:

- must be very cheap to build and maintain,
- do not need to operate quickly,
- must function properly for decades, without central control and resist interference from attackers or catastrophic failures of storage media such as fire or theft.

These features lead to some design principles:

Cheap storage is unreliable: Non-cheap storage is reliable in our time. Replicas can guarantee that data won't be lost at once.

No long-term secrets: Long-term secrets are vulnerable since require storage that is effectively impossible to replicate, audit, repair or regenerate.

Use inertia: There is never a need for rapid change in a long-term preservation system. Some change is inevitable but whenever a repair has to be done, the system has no need for speed to make it.

Avoid third-party decomposition: Digital preservation system should avoid that an attacker can "cash in" a history of good behavior if evidence of past good behavior is accumulated by the system.

Reduce predictability: Attackers can predict the actions of peers that keep the same behavior to choose tactics.

Intrusion detection: It is important that system provides intrinsic intrusion detection by attackers and generates an alarm.

Assume a strong adversary.

The LOCKSS design follows theses design principles. Applying those design principle LOCKSS avoid that attackers can make irrecoverable damages in to it without being detected.

LOCKSS System

LOCKSS system is a long-term large-scale application that models the physical document system into a Web-published document system.

The Digital Preservation system LOCKSS (Lots of Copies Keep Stuff Safe) allows libraries that run large number of independent, low-cost, persistent web caches to:

1. collect by crawling the journal web-sites to pre-load themselves with newly published material,
2. distribute by acting as a limited proxy cache for the library's local readers, supplying the publisher's copy if it is available and the local copy otherwise,
3. preserve by cooperating with other caches that hold the same material to detect and repair damage. Web-caches cooperate by participating in "opinion polls" protocol and voting on their cached documents.

Using the voting protocol peers agree or disagree with each other on the material. This procedure is the best available guarantee that content is authentic and correctly preserved.

Each peer in a network below LOCKSS holds a huge set of Archival Units (AU), which describes its content. Because each peer holds a different set of AUs, the voting protocol treats each AU independently.

Peers call polls on an AU expecting to obtain votes from other peers on that AU. Peers can contribute with agreeing or disagreeing vote on an AU. A peer loses a poll on an AU when it gets a quantity of disagreeing votes bigger than a minimum number pre-assumed by the system. In that case the AU is corrupt and the peer must request a repair to other peers that offer it a good copy, in the same way they would for local readers.

The Opinion Poll Protocol

The protocol supports two roles for participating peers:

1. the poll initiator calls polls on its own Aus expecting to get vote on it,
2. the poll participant or voter is a peer who is invited into the poll by the poll initiator and who votes if it has the necessary resources. A voter need not find out the result of a poll in which it votes.

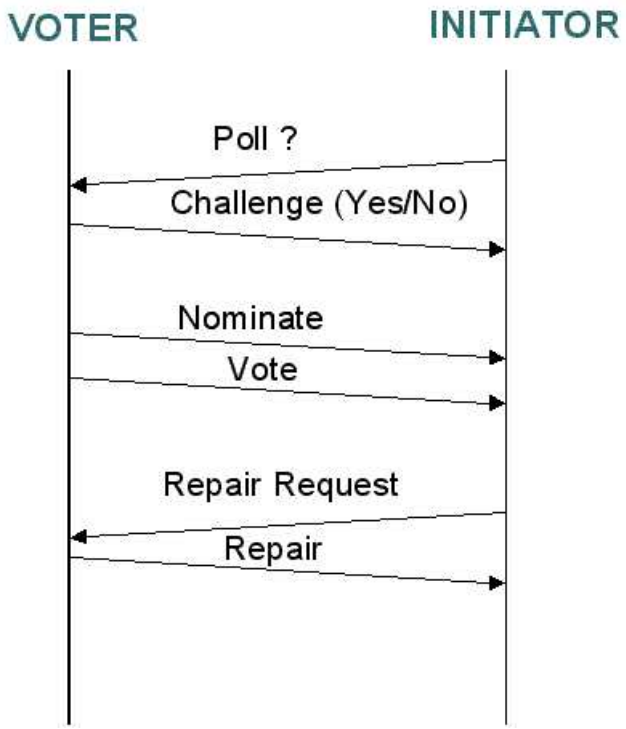

Figure 1.

We consider that peers preserve a copy of a single AU, obtained from a publisher who is no longer available.

The poll initiator invites into its poll a small subset of the peers, hoping they will offer votes on their version of the AU. Invited peers (voters) with had accepted the invitation to the poll computes a hash on their own AU, which is returned in a vote.

If the pool initiator gets more votes that a pre-assumed minimum number that disagree with its own version of the AU, it request a repair on its AU by fetching the copy of a voter who disagreed, and re-evaluates the votes. If the result of the poll justifies neither a corrupt copy nor a good copy of an AU (an inconclusive poll), then the caller raises an alarm to attract human attention to the situation.

A LOCKSS peer calls opinion polls on the contents of an AU it holds random time. Opinion polls must be called at a rate greater than any anticipated rate of random damage made by an attacker.

Communication between peers is encrypted via symmetric session keys, derived using Diffie-Hellman key exchanges[2]. During the communication, the protocol requires both poll initiators and voters to expend provable computational effort[5] in amounts related to underlying system operations (hashing of an AU), as a means of limiting Sybil attacks[3].

A peer participating in the LOCKSS protocol shall have a good intent (benign peer) or be an attacker with malicious intent, we make the following distinctions between different types of peers that can participate in the LOCKSS protocol:

- A malign peer is part of a conspiracy of peers attempting to subvert the system.
- A loyal peer is a non-malign peer, i.e., one that follows the LOCKSS protocol at all times.
    - A damaged peer is a loyal peer with a damaged AU.
    - A healthy peer is a loyal peer with the correct AU.

The protocol goal is to guarantee that loyal peers remain in the healthy state and even a powerful adversary cannot damage a significant proportion of the loyal peers without detection.

Protocol Description

Each peer in the LOCKSS maintains two peer lists for every AU it holds:

- Reference list - contains information about other LOCKSS peers it has recently encountered. Reference list entries have the form [peer IP address, time inserted].
- Friends list - contains information about peers with whose operators or organizations the peer has an out-of-band relationship. Friend list entries have the form [peer IP address].

A peer maintains for every AU it holds a poll counter that records the number of polls the peer has called on that AU since first obtaining it.

Bootstrapping: When a peer first enters a LOCKSS network for a given AU, or when it reinitializes after a failure, it copies all entries from its current friends list into its reference list.

Poll Initiation: Initially, a peer creates its inner circle list by coping a fixed number random peers from its reference list to it.

Then the initiator computes a fresh, random poll identifier and for each inner circle peer, the poll initiator chooses a fresh, random Diffie-Hellman public key. Afterward, it invites peers from its inner list to a poll by sending a poll message, of the form [Poll ID, DH Public Key].

The initiator waits for PollChallenge messages from the invited inner circle peers and sets a challenge timer to stop waiting.

The initiator removes from its inner circle list those peers:

- who respond with a negative PollChallenge message,
- who do not respond by the time the challenge timer expires,
- from whom the initiator receives multiple PollChallenge messages with conflicting contents(may be the victim of a spoofer).

If the initiator ends up with fewer inner circle peers than the minimum required, it invites additional peers into the poll via more Poll messages, or aborts the poll if it has no more peers in its reference list.

Poll Challenge: After receiving a Poll message from a poll initiator, a LOCKSS peer chooses a fresh, random challenge value, a fresh, random Diffie-Hellman public key, and computes a symmetric session key from it and from the poll initiator's public key included in the Poll message.

It sends back a PollChallenge message of the form [Poll ID, DH Public Key, challenge, YES/NO] indicating whether it declines or accepts the poll's invitation.

Finally, the peer waits for a Poll-Proof message from the poll initiator and sets an effort timer. If the message never arrives, the peer discards all poll state. Otherwise, the peer verifies the PollProof message.

Poll Effort Proof: The initiator computes the provable effort for its poll invitation for all inner circle peers it had got a valid, affirmative challenge message and send it by Proof message. The computation of the provable effort depends on the challenge value from the challenge message.

The initiator also sends Proof messages to poll participants who responded to the initial invitation with a negative PollChallenge but it does not computer the provable effort, though it uses a random value.

Provable effort is based on a class of memory-bound functions[1] (MBF) proposed by Dwork et al.[4] to prevent email spam. These cryptographic functions have a computation phase, yielding a short proof of effort, and a verification phase, which checks the validity of that proof. A parameter of the system sets the asymmetry factor by which the computation phase is more time consuming than the adjustable cost of the verification phase[6].

The Proof message is encrypted using the session key and is of the form [Poll ID, poll effort proof].

Pool Nomination: A voter gets the PollProof message and verifies the poll effort proof. If the verification succeeds, the voter chooses some peers at random from its own reference list, send them via a Nominate message of the form [Poll ID, Nominations] encrypted with the session key.

Outer Circle Invitations: Based on the Nominate messages returned from voters, the initiator discovers new peers that maintain the same AU.

From the nominations list, the initiator creates its outer circle list with those new peers removing from every nomination list peers already contained in its reference list, and then it chooses an equal number of peers from every nomination list at random for its outer circle list.

The initiator invites outer circle peers into the poll but nominate messages and vote from outer circle participants are ignored.

Finally, the initiator waits vote and it sets a vote timer.

Poll Vote: The voter sends back an encrypted Vote message to the poll initiator. The vote message contains a hash of the AU interleaved with provable computational effort.

The Vote message construction is divided into rounds. The first round depends on the challenge value and poll identifier to compute a proof of computational effort and the voter hashes this proof with a portion of the AU. Subsequent rounds depend on the output of the previous round to compute a new proof of computational effort and the voter hashes this proof with a new portion of the AU. The output of all round forms the vote message.

Vote Verification: Vote can be classified in: invalid, valid disagreeing with the initiator's AU, agreeing with the initiator's AU.

Because votes are constructed in rounds, they are also verified in rounds. For each roung, the initiator verifies the proof of computational effort included in the Vote message for the corresponding voting round.

If the proof is incorrect, the initiator deems the vote invalid and verification stops. Peers that supply an invalid vote are removed from the pool (inner or outer circle), and from the initiator's reference list. Invalid votes indicate fault or malice because they are not consistence with the other previous messages from the protocol.

If the proof is correct, the initiator deems the vote valid and the initiator hashes the proof with the corresponding portion of its own copy of the AU to compare with the hash the proof with the corresponding portion of the voter's copy of the AU got from the vote message. If the result does not match the hash in the vote, the vote is declared disagreeing. Otherwise, it is declared agreeing.

After verifying all received Vote messages, the initiator analyzes the results to determine whether its AU replica is correct.

If the initiator gets a number of valid votes smaller to the quorum pre- determined it does not make a decision on its AU, though it updates its reference list, and immediately calls another poll. If a peer as not able to conclude a poll in an AU for a long time, the initiator raises an alarm.

The initiator decision acts as follows: Agreeing votes are no more than a pre-determined number D. The initiator considers its current AU copy damaged and repairs it. Agreeing votes are at least a pre-determined value E. This is the only way in which an opinion poll concludes successfully. The initiator updates its reference list and schedules another poll at a random future time. Agreeing votes are more than D but fewer than E. The initiator considers the poll inconclusive and raises an alarm.

Repair Solicitation: When the initiator decides that its AU is damaged, it picks at random one of the disagreeing inner circle voters and sends it an encrypted RepairRequest message containing the poll identifier.

Repair: The Repair message contains the poll identifier and its own copy of the AU, encrypted with the symmetric session key.

When the initiator receives a Repair message, it re-verifies any disagreeing votes given the new AU and re-decided the results. If a peer supply a repair that is inconsistency between a vote and the AU on which that vote was purportedly computed, the initiator removes the supplier from its reference list since it may signal a fault at the repair supplier.

Protocoll Analysis

LOCKSS system must avoid the adversary from gaining a foothold in a poll Initiator's reference list, make it expensive for the adversary to waste another peer's resources, and detected the adversary's attack before it progresses far enough to cause irrecoverable damage to defend itself against attackers.

For that reason, LOOCKS uses effort sizing, rate limiting, reference list churning, and obfuscation of protocol state to make it expensive and slow for an adversary to gain a significant foothold in a peer's reference list or waste other peers resources and it raises alarms whenever detects intrusion of attack.

Effort Sizing

To ensure that large changes to the system require large efforts, LOCKSS adjust the amount of effort involved in message exchanges for voting, discovery, and poll initiation, by embedding extra, otherwise unnecessary effort.

In poll initiation, an initiator must expend effort to avoid that a malign peer would initiate spurious polls at no cost, causing loyal peers to waste their resources.

In voting, the cost of constructing a vote must be greater than the cost of processing the vote to avoid that a malign peer could vote more than one time.

In discovery, peers from outer list must first participate in a whole poll and generate a valid agreeing, which means that they must prove substantial effort, before getting in the initiator's reference list. This makes it expensive and time-consuming for a malign peer to get an opportunity to vote maliciously and effectively.

Timeliness of Effort

To avoid third-party reputation, LOCKSS guarantees that only proofs of recent effort can affect the system.

The poll effort proof from a Proof message depends on a challenge value from the challenge message and the first round of a subsequent vote depends on that proof, and each subsequent round depends on the round preceding it.

Rate Limiting

To avoid that the system changes rapidly no matter how much effort is applied to it, LOCKSS uses rate-limiting techniques.

An adversary power of damage is rated by a other peer invitation to a poll. The adversary has the opportunity to damage the system when a loyal peer calls it to a poll.

Reference List Churning

It is important for a peer to avoid depending on a fixed set of peers for maintenance of its AU, because those peers may become faulty or subversion targets.

Every time a poll is concluded, the LOCKSS peer updates its reference list. The update consists in churning into its reference list a few peers from its friends list in addition to the outer circle-agreeing voters. The initiator expects not to fill its reference list with malign conspirators.

Obfuscation of Protocol State

Since a powerful adversary is assumed, the protocol state is obfuscated in two ways to deny adversary information about a poll.

- All protocol message exchanged encrypt but the first using a fresh symmetric key for each poll and voter.
- All peers invited into a poll, even those who decline to vote, go through the motions of the protocol, behind the cover of encryption.

This prevents an adversary from using traffic analysis to infer state such as the number of loyal peers who actually vote in a specific poll.

Intrusion Detection (Alarms)

The protocol raises an alarm when a peer:

1. determines that a poll is inconclusive,
2. suspects local spoofing,

3. has been unable to complete a poll for a long time.

Adversary

LOCKSS assume that attacks come from powerful adversaries. The adversary can attack in different way depending on his goal: get a good copy of a AU, subvert loyal peers, replace the good content of a peer with a bad version and so on.

Theft: The adversary wishes to obtain published content without the consent of the publisher. LOCKSS protocol allows that a peer supplies a transfer of an AU (repair) only if the requester has previously proved with an agreeing vote that it once had the same content. The protocol cannot be used to obtain a first instance of an AU.

Free-loading: The adversary wishes to obtain services without supplying services to other peers in return. In LOCKSS protocol the content of an AU is only available to peers who prove through voting that they have had the content in the past allowing no free-loading.

Stealth Modification: The adversary wishes to replace the good content with his version (the bad content). His goal is to change, through protocol exchanges, as many replicas of the content held by loyal peers as possible without being detected, i.e., before the system raises an alarm. To achieve this goal, he must repeatedly find a healthy poll initiator, convince it that it has a damaged AU replica without causing it to raise any alarms, and then, if asked, conveniently offer it a repair with the bad version of the AU. The adversary first tries to build a foothold in loyal peers' reference lists and then he attacks. LOCKSS defends against it using rate-limiting principle, alarms and reference list churning.

Results

The results were based on the effects on LOCKSS of an attack by an adversary following the stealth modification strategy.

This strategy is divide in two phases:

1. The adversary seeks only to extend his foothold in loyal peers' reference lists.

2. Malign peers seek to change the loyal peers' replicas of the AU with the bad that the adversary wishes to install throughout all peers.

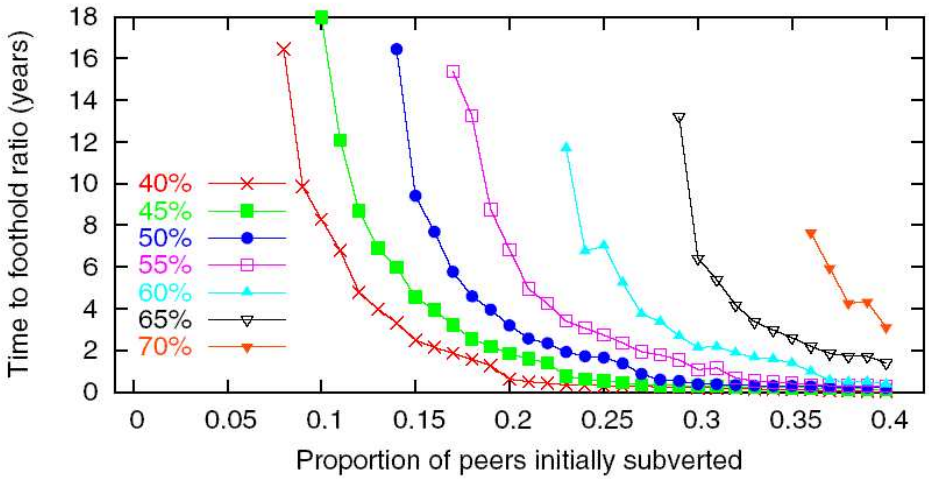

Figure 2. Minimum time for the stealth adversary to achieve various foothold ratios, starting with various proportions of initially subverted peers.

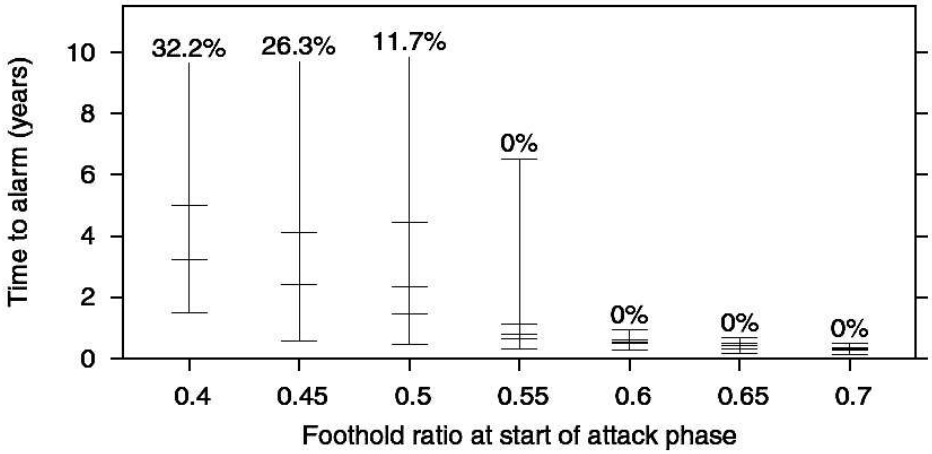

Figure 3. The time from the start of the attack phase (in the stealth strategy) to the time of detection, for different starting reference list foothold ratios. Ticks split the value distributions into quartiles. Percentages above the distributions indicate runs that did not generate an alarm.

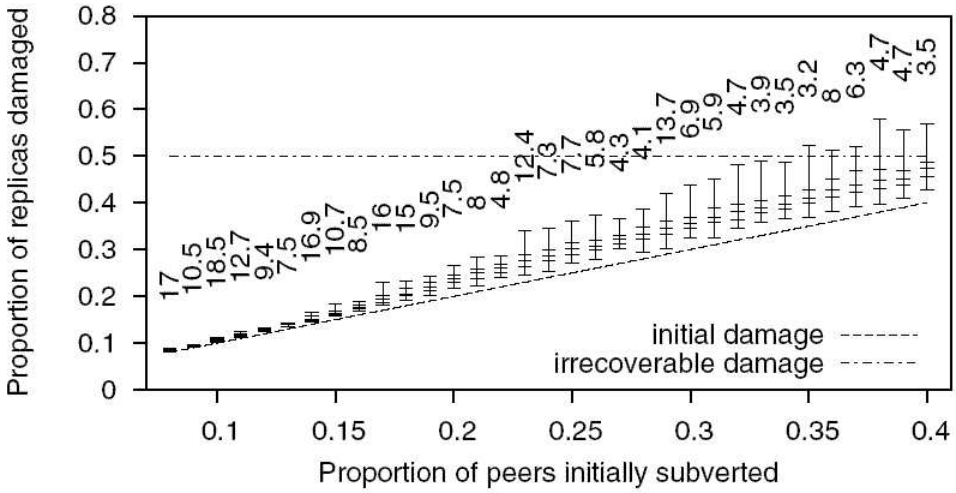

Figure 4. The percentage of bad replicas as a function of initial subversion. Ticks split the value distributions into quartiles. Numbers above the distributions show the time in years needed by the adversary to cause maximum damage. The diagonal line shows the damage due to peer subversion. The horizontal line shows the threshold for irrecoverable damage.

The results show that the probability of a stealth adversary causes irrecoverable damage remains very low even for an initial subversion of 1/3, and then increases gradually (overs decades). Irrecoverable damage occurs when the damaged (loyal) peers form more than 50% of the population, although the probability of irrecoverable damage is very low because of the mecanism of intrusin detection.

Conclusion

LOCKSS is a peer-to-peer digital preservation system that models the physical document system into a Web-published document system providing a protocol that allows libraries (web-caches) to collect the material to which they subscribe, distribute it to local readers and preserve it by cooperating with others libraries.

LOCKSS combines massive replication, rate limitation, inherent intrusion detection and costly operations to resist attacks by some extraordinarily powerful adversaries over decades.

LOCKSS follows the design principles for digital preservation system using replicas across peers as persistent storage, no long-term secrets, providing inherently expensive operations and limiting the rate of possible change in the system to avoid rapid changes in the system, make operations time- consuming so attackers do not devastate the system quickly, requiring substantial recent effort, reducing predictability by churning the reference list and using alarms to detect intrusion.

Bibliography

[1] Mart Abadi, Mike Burrows, Mark Manasse, and Ted Wobber. Moderately hard, memory-bound functions.

[2] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. IEEE Transactions on Information Theory, IT-22(6):644–654, 1976.

[3] J. Douceur. The sybil attack, 2002.

[4] C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam, 2002.

[5] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In CRYPTO '92:Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology, pages 139–147. Springer-Verlag, 1993.

[6] P. Maniatis, M. Roussopoulos, T. Giuli, D. Rosenthal, M. Baker, and Y. Muliadi. Preserving peer replicas by rate-limited sampled voting, 2003.