Search | Back Issues | Author Index | Title Index | Contents

**A R T I C L E S**

# LOCKSS: A Permanent Web Publishing and Access System

Vicky Reich
Stanford University Libraries
*vreich@stanford.edu*

David S. H. Rosenthal
Sun Microsystems Laboratories

LOCKSS: <http://lockss.stanford.edu/>

## Abstract

LOCKSS (Lots Of Copies Keep Stuff Safe) is a tool designed for libraries to use to ensure their community's continued access to web-published scientific journals. LOCKSS allows libraries to take custody of the material to which they subscribe, in the same way they do for paper, and to preserve it. By preserving it they ensure that, for their community, links and searches continue to resolve to the published material even if it is no longer available from the publisher. Think of it as the digital equivalent of stacks where an authoritative copy of material is always available rather than the digital equivalent of an archive.

LOCKSS allows libraries to run web caches for specific journals. These caches collect content as it is published and are never flushed. They cooperate in a peer-to-peer network to detect and repair damaged or missing pages. The caches run on generic PC hardware using open-source software and require almost no skilled administration, making the cost of preserving a journal manageable.

LOCKSS is currently being tested at 40+ libraries worldwide with the support of 30+ publishers.

## The Problem

The web is an effective publishing medium (data sets, dynamic lists of citing papers, e-mail notification of citing papers, hyperlinks, searching). Increasingly, web editions are the "version of record" and paper editions of the same titles are merely a subset of the peer reviewed scholarly discourse. Scientists, librarians, and publishers are concerned that this important digital material, the record of science, will prove as evanescent as the rest of the web. In addition, each of these communities has specific needs:

- Future generations of scientists need access to this literature for research, teaching, and learning.

- Current and future librarians need an inexpensive, robust mechanism, which they control, to ensure their communities maintain long-term access to this essential literature.

- Current and future publishers need assurances that their journals' editorial values and brands will be available only to authorized and authenticated readers.

Therefore, the problem is to preserve an authorized reader's access to the web editions of scientific journals while staying within libraries' budgets and yet respecting publishers' rights.

## Requirements

Technically, any solution must satisfy three requirements:

- The content must be preserved as bits;
- Access to the bits must be preserved;
- The ability to parse and understand the bits must be preserved.

There is no single solution to this problem and, furthermore, having only a single solution would mean that materials would still be vulnerable to loss or destruction. Diversity is essential to successful preservation. By proposing LOCKSS we are not discounting other digital preservation solutions; other solutions must also be developed and deployed.

In particular, we believe that there are needs in this area for both centralized "archives" and a distributed library system like LOCKSS. For digital materials, the terms *archiving*, *library*, and *preservation* tend to be used interchangeably. For our purposes, we follow the dictionary in defining an *archive* to be a place where public records are

stored and a *library* to be a place where people read and/or study materials. *Preservation* is the action of keeping materials from injury or destruction. Both library and archive materials need preservation.

## Models of Repositories

There are two models of digital preservation and archiving repositories: centralized and decentralized. A key question to ask of each model is "what are the costs of preserving different types of materials and on whom do these costs fall?"

The *centralized* model envisages a small number of tightly controlled repositories. Each repository a) does the entire job of preserving content, b) requires expensive hardware, and c) requires sophisticated technical staff. To establish a centralized system, publishers and librarians must take legal and data management actions cooperatively. Preservation costs are borne by a few. Centralized systems are focused on preserving bits and pay less attention to ensuring access. Indeed many of these systems are explicitly "dark" archives, in which content will not be accessed until some "trigger" event (migration, publisher failure, etc.) occurs.

The *decentralized* model envisages a large number of loosely controlled repositories. Each repository or node in the system a) does some but not the whole job of preserving content, b) uses relatively inexpensive hardware, and c) needs relatively little technical expertise to maintain the hardware and software. The content at each repository is in constant use, under constant scrutiny, and undergoing continual repair. In a decentralized system, publishers take little or no action to preserve the content they publish; librarians take action to preserve access for their local communities. While libraries bear the costs of digital preservation, each participating library bears only a small fraction of the total cost, in proportion to its resources and priorities. The benefits accrue only to the participating library's communities. Decentralized systems are focused on preserving access rather than just preserving the bits. These systems count on the redundancy inherent in distributed systems to keep the bits safe.

## Overview of LOCKSS

LOCKSS is a digital preservation Internet
appliance, not an archive. Archives exist to
maintain hard-to-replicate materials, and
access is sacrificed to ensure preservation.
LOCKSSS is more akin to a global library
system. Libraries hold fairly common
materials in "general collections" with access
as the primary goal. A key difference between
LOCKSS and "general library collections" is
that the action of preserving material in the
collection is intertwined with the provision of
access to the end user.

Librarians retain paper publications to ensure long-term accessibility.
One could visualize all the libraries in the world as parts of a system -- a
very informal, highly decentralized, highly replicated system. The
primary goal of this system is to provide access to material, but
providing access in this way also ensures that documents are not lost as
a result of publisher takeovers, malicious actions, natural disasters, or
official edicts. Generally, someone at a local library will find it easy to
access the paper copy of a particular book or journal. Once a book or
journal has been published it is hard to "unpublish" it by finding and
destroying all copies.

LOCKSS is modeled on this paper system. With the LOCKSS model,
libraries run persistent web caches. Their readers can use these caches
to access the journals to which the library subscribes, whether or not
the journals are still available from the publisher.

The system makes it easy to find a copy of an article but hard to find all
the copies of the aricle, thus making it hard to "unpublish" it. Very
slowly, the LOCKSS caches "talk to each other" to detect missing or
damaged content. If a cache needs to repair content, it can get a
replacement from the publisher or from one of the other caches via
"inter-library loan".

### How the Data Flows
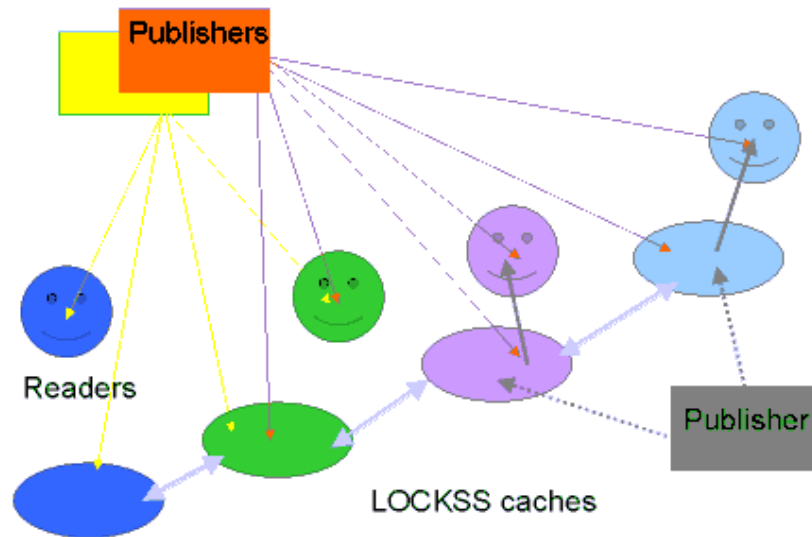
Data flows – an approximation

**Figure 1: In this example, each LOCKSS cache (oval) collects journal content from the publisher's web site as it is published. Readers (circles) can get content from the publisher site. When the publisher's web site is not available (gray) to a local community, readers from that community get content from their local institution's cache. The caches "talk" to each other to maintain the content's integrity over time**.

### Reader's Perspective

LOCKSS' key goal is to preserve a reader's access to content published on the web. Readers expect two kinds of access. They expect that:

- When they click on a link to it, or type in a URL, the relevant page will be delivered with minimal delay and no further interaction.

- When they enter terms into a search engine that should match the relevant page, it will be among the returned matches.

Readers who use the web are learning that if a link doesn't resolve to a page, or a search engine can't find a page, further attempts to find the information the page carries are unlikely to be worth the effort. This poses problems for those who use preservation techniques that concentrate on preserving bits; the bits may be preserved yet the reader may not know how to access them, or even that the preserved bits exist.

In contrast, LOCKSS focuses on preserving the service of having links resolve to, or searches to find, the relevant content. An institution

using LOCKSS to preserve access to a journal in effect runs a web cache devoted to that journal. Readers use the cache as a proxy in the normal way. At intervals the cache crawls the journal publisher's web site and pre-loads itself with newly published (but not yet read) content. Just as other types of caches are invisible to their users, so is LOCKSS. The LOCKSS cache transparently supplies pages it is preserving even if those pages are no longer available from the original publisher's web site.

An institution can include the contents of the cache among the pages indexed by its local search engine, and provide its readers with searching across all the journals to which it subscribes. At present, readers typically have to search individual collections of journals separately.

**Librarian's Perspective**

Librarians subscribe to journals on behalf of their readers in order to provide both immediate and long-term access. With the advent of the web, for the most part libraries are forced to lease rather than own the web-based content. Leasing provides immediate access but carries no guarantee of long-term access. Some journals provide their peer reviewed content through off line storage media (tape, CD-ROM, paper), but then links don't resolve and searching is harder to accomplish. A major flaw with web publishing is that there has been no mechanism to implement the traditional purchase and own library model.

A major flaw with web publishing is that there has been no mechanism to implement the traditional purchase-and-own library model. LOCKSS is changing this by demonstrating that it is both easy and affordable to operate a purchase model for web journals. The subscribing library bears costs analogous to the costs of putting paper copies on shelves, keeping track of them and lending or copying them as needed. A library using LOCKSS to preserve access to a collection of journals pays for the equipment and staff time to run and manage a cache containing the full content of the journals. Unlike normal caches, the LOCKSS cache is never flushed and, over the long term, the full content remains accessible.

Because individual libraries must pay for the preservation of the content to which they subscribe, it is essential that the price they pay be as low as possible. LOCKSS is free, open-source software designed to run on inexpensive hardware. The machines the LOCKSS team is using for the beta test cost less than $800 each and each machine is capable of storing the content of 5 years worth of a major journal's issues. Running LOCKSS requires so little staff time that one alpha test site

complained they learned nothing about the system over the course of 10 months while running it. LOCKSS' low cost and democratic structure (each copy is as valuable as any other) empowers smaller institutions to take part in the process of digital preservation.

In normal operation, an ordinary cache will only act as a proxy for, and thus supply content to, the host institution's own readers. But in a rough analog of inter-library loan, LOCKSS caches cooperate to detect and repair damage. If damage to a page is detected, the LOCKSS cache fetches a copy of the page from the publisher or from another cache. A LOCKSS cache will only supply a page to another LOCKSS cache if the requesting cache at some time in the past proved that it had the requested page. In this way, LOCKSS prevents freeloading. Those who contribute to the preservation of the journal are rewarded with continued access; those who do not contribute to the journal's preservation are not provided with replacement pages.

**Publisher's Perspective**

Publishers want to maintain journal brand and image. They want material available for future society members and other subscribers. Most publishers will save money and serve their readers better if the transition to electronic-only journals can be completed. They want to encourage libraries to purchase and/or activate online versions of journals. One major obstacle to libraries purchasing online journals is resistance to the rental model with its lack of credible assurance of long-term access. Publishers are unhappy with a purchase model for electronic journals because:

- They fear the journal content will be illegally replicated, or leaked, on a massive scale once copies are in the custody of others; they want their access control methods enforced.

- They want to retain access to reader usage data and have access to the record of the reader's interactions with their site.

LOCKSS solves the reader's and the librarian's problems. It enables librarians to collaborate to preserve readers' access to the content to which they subscribe, but it also addresses the publisher's concerns.

- Because content is provided to other caches only to repair damage to content they previously held, no new leakage paths are introduced.

- Because the reader is supplied preferentially from the publisher, with the cache only as a fallback, the publisher sees the same interactions they would have seen without LOCKSS.

LOCKSS has other advantages from the publisher's perspective:

- It returns the responsibility for long-term preservation, and the corresponding costs, to the librarians. Although publishers have an interest in long-term preservation, they cannot do a credible job of it themselves. Failures or changes in policy by publishers are the events librarians are most interested in surviving.

- Publishers could run LOCKSS caches for their own journals and, by doing so, over time could audit the other caches of their journals. A non-subscriber cache would eventually reveal itself by taking part in the damage detection and repair protocol. The mere possibility of detection should deter non-subscribers from taking part in LOCKSS. Just as the publisher cannot be sure he has found all the caches, the caches cannot be sure none of the other caches belongs to the publisher.

## How Does LOCKSS Preserve Content?

LOCKSS has two tasks in preserving content:

- It needs to detect, and if possible repair, any damage that occurs through hardware failure, carelessness or hostile action.

- It must also detect, and if possible render ineffective, any attacks.

### Detecting and Repairing Damage

Damage to contents is a normal part of the long-term operation of a digital storage system. Disks fail, software has bugs, and humans make mistakes. To detect damage, the caches holding a given part of a journal's site vote at intervals on its content. They do so by calculating digital hashes of the content and running polls on the values of these hashes:

- In the absence of damage the hashes will agree.

- If they disagree, one of the losers calls a sequence of polls to walk down the tree of directories to locate the damaged files.

- When a damaged file is located, a new copy is fetched to replace it.

- If the file is not available from the publisher, it will be requested from one of the winning caches.

- If a cache receives a request for a page from another cache, it examines its memory of agreeing votes to see if the requester once

agreed with it about the page in question. If the requester did, a new copy will be supplied.

LOCKSS polls are not like the elections of conventional fault-tolerant systems in which voting is mandatory. They are like opinion polls, in which only a sample of the potential electors takes part. Only a sample of the LOCKSS caches holding given content vote in any one poll. Note that even caches that don't vote hear the votes of those that do. They can decide whether they agree or disagree with the majority in the poll. Normally there will be no damage and each of the polls will be unanimous. If there is a small amount of random, uncoordinated damage each poll will be a landslide, with only a few disagreeing votes. In normal landslide polls, the majority of systems will be reassured that their copy is good without voting.

Details of this system were presented to the 2000 Usenix conference (http://lockss.stanford.edu/freenix2000/freenix2000.html).

### Hampering the "Bad Guy"

If there is ever a poll whose result is close it is highly likely that a "bad guy" is trying to subvert the system. Attempts at subversion are endemic in peer-to-peer systems like LOCKSS. (See http://www.wired.com/news/infostructure/0,1377,41838,00.html.) A "bad guy's" goal might be to change the consensus about some content in the system without being detected.

- A "bad guy" who infiltrated only a few caches and made matching changes to each of their contents would appear to be random damage. The other caches would not change their contents to match.

- If the "bad guy" infiltrated a substantial number of caches, even a small majority, he would cause polls whose results were close. In the event that the material was no longer available from the publisher, the caches which had not been subverted might replace their good copies with the "bad guy's" modified ones. However, the close results of the polls would alert the system's operators that something was wrong.

- Only if the "bad guy" infiltrated the overwhelming majority of caches would his change be both effective and undetected. The remaining "good" caches would appear to have been damaged and would fetch the "bad guy's" versions.

Another obstacle for the "bad guy" is that LOCKSS is designed to run extremely slowly. A single poll may take days. It takes many polls to

have an effect. By preventing the system from doing anything quickly, we prevent it doing what the "bad guy" wants. Because it is difficult for the "bad guy" to operate without raising an alarm, it is likely that the administrators of the system would notice and react to attempts at subversion while those attempts were underway. The inter-cache communication can run very slowly because it does not delay readers' accesses to the journals, which are purely local operations.

The difficulty in infiltrating an overwhelming majority of caches lies in being sure you have found enough of them. If there are a lot of caches it will be a long time between votes from any one of them. The set of caches holding the material to be attacked will be changing over a somewhat slower time-scale as caches die and are born. Because the underlying communication protocol is inherently unreliable, some caches will not hear some votes. These uncertainties work against the "bad guy".

### Reputation System

To make life even harder for the "bad guy", each LOCKSS cache maintains a record of the behavior of the other caches in the form of a reputation. As caches are observed taking good actions, their reputation is enhanced; as they are observed taking bad actions, their reputation is degraded. When a cache tallies a poll, it will take action only if the average reputation of those voting on the winning side is high enough. This has three effects:

- A "bad guy" can only have an impact by behaving well for long enough to build up a good reputation. If the "bad guy" has to spend most of his time acting as a "good guy" the system may get sufficient benefit from the good actions to outweigh the bad ones.

- The "bad guy" must achieve his nefarious aims quickly, before his reputation is eroded far enough to render him ineffective. But the system is designed to run very slowly.

- The "bad guy" might find ways to skew the sample to include his co-conspirators and exclude the "good guys". However, unless the conspiracy overwhelms the "good guys" very quickly, the appearance of "bad guys" acting in concert will damage their reputations even faster than if they were acting alone.

More detail on the struggle between good and evil in the LOCKSS context can be found at http://lockss.stanford.edu/locksssecurity.html.

## Running LOCKSS

The current LOCKSS version runs on generic PCs. At current prices, a suitable machine with a 60GB disk in a 1U rack-mount case should cost about $750. The system is distributed as a bootable floppy disk. The system boots and runs Linux from this floppy; there is no operating system installed on the hard disk. The first time the system boots it asks a few questions, then writes the resulting configuration to the floppy, which is then write-locked. At any time, the system can be returned to a known-good state by rebooting it from this write-locked disk.

Each time the system is booted, it downloads, verifies and installs the necessary application software, including the daemon that manages the LOCKSS cache and the Java virtual machine needed to run it. The system then runs the daemon and starts the HTTP servers that provide the user interface web pages. The cache's administrator can use these pages to specify the journal volumes to cache and monitor the system's behavior.

## Project Status

Up-to-date project status is available at http://lockss.stanford.edu /projectstatus.htm.

### Alpha Test

Design and development of LOCKSS started in 1999. Testing started on 6 old 100MHz PCs late that year. An "alpha" version of the software, without a user interface or any precautions against the "bad guy", ran from May 2000 through March 2001 with around 15 caches. The test content was about 160MB representing three months of AAAS *Science Online*. Alpha sites were Stanford University, the University of California, Berkeley, the Los Alamos National Laboratory (LANL), the University of Tennessee, Harvard University, and Columbia University. This test established that the basic mechanisms worked. The system was able to collect the test content and repair both deliberate and accidental damage to it. The system survived a fire at LANL, network disruptions at Stanford, relocation of the machine at Berkeley, and flaky hardware at Columbia.

### Beta Test

The worldwide "beta" test began in April 2001, using an almost complete implementation of the system. Approximately 35 publishers are endorsing the test. Over 40 libraries, with about 60 widely distributed and varyingly configured caches, have signed on to the project. They include major institutions, such as the Library of

Congress and the British Library, and smaller institutions, such as the University of Otago in New Zealand.

Beta will test the usability and performance of LOCKSS, measure its impact on network and web-server traffic, and provide some estimates of the costs of running an individual cache and the system as a whole. The test content is a total of about 15 GB from *BMJ*, *JBC*, *PNAS* and *Science Online*. The test content is provided to the caches by shadow servers, which partially mirror the publishers' web sites. They isolate the LOCKSS data streams and allow us to simulate journal failures. We hope in the later stages of beta to add other publishers' journals, and other types of content such as government documents.

## Acknowledgements

L O T S   O F   C O P I E S   K E E P   S T U F F   S A F E

*(29 June 2001, a correction was made to Vicky Reich's email address.)*

---

**Top** | **Contents**
**Search** | **Author Index** | **Title Index** | **Back Issues**
**Previous Article** | **In Brief**
**Home** | **E-mail the Editor**

---

**D-Lib Magazine Access Terms and Conditions**