Authenticity and Availability in PIPE Networks

Brian F. Cooper, Mayank Bawa, Neil Daswani, Sergio Marti and Hector Garcia-Molina

Department of Computer Science
Stanford University
{cooperb,bawa,daswani,smarti,hector}@db.stanford.edu

Abstract

We describe a system, which we call a Peer-to-peer Information Preservation and Exchange (PIPE) network, for protecting digital data collections from failure. A significant challenge in such networks is ensuring that documents are replicated and accessible despite malicious sites which may delete data, refuse to serve data, or serve an altered version of the data. We enumerate the services of PIPE networks, discuss a threat model for malicious sites, and propose basic solutions for managing these malicious sites. The basic solutions are inefficient, but demonstrate that a secure system can be built. We also sketch ways to improve efficiency.

Key words: digital preservation, malicious sites, failures

1 Introduction

For centuries, librarians and archivists have studied methods for preserving paper library materials despite acidic paper, humidity and broken bindings. However, the prevalence of digital collections means that preservation must be reconsidered. Hard drives may crash, users may accidentally delete data, publishers may go out of business and shut down servers that are providing digital materials. In a cooperative environment, such as a grid computing infrastructure, loss of information can cause more than a temporary inconvenience to one user; it may mean the loss of vital, difficult to reproduce scientific results, as well as the interruption of important computations that depend on this data. Similarly, in a distributed digital library, loss of digital documents can mean permanent loss of valuable cultural artifacts.

To avoid the loss of critical materials, data must be replicated at multiple sites. After a site failure, data can be restored from another site. Several systems have been built on this principle, including OceanStore [12], LOCKSS [22] and SAV [5,6]. However, the replication solution introduces a new set of challenges: how to ensure that the backup sites can be trusted to produce authentic copies of the stored data. For example, a scientist, Jane, may store information about her experiments at a remote site that employs another scientist, Joe, who is performing similar experiments. If Joe is unethical, he may want to prevent Jane from retrieving her data, so that he can claim the work as his own or at least slow down Jane's progress. Joe may destroy Jane's data, or modify it so that it contains incorrect information. Joe may also reprogram his site's server to refuse requests to serve the data. In any event, if Jane's site has a failure and she must retrieve the information from Joe's site, she will be stymied by Joe's malicious actions. As another example, people that disagree with a particular document, such as "The Origin of Species," may try to delete or alter all copies of that document.

An archival system must be designed to prevent, detect, manage and/or recover from these malicious behaviors, so that the preservation infrastructure can provide services to its users. Similarly, a complete system must ensure that important data remains preserved even if the creator or publisher leaves the system, that users can verify the authenticity of retrieved documents, that agreements remain valid years after they are originally concluded, and so on.

Our approach to dealing with these problems is to develop a reliable preservation service built on a peer-to-peer architecture. Such a *Peer-to-peer Information Preservation and Exchange* (PIPE) service would serve as a reliable substrate on top of which other distributed grid and digital library applications could be built. Building a replication system on a peer-to-peer architecture has several advantages, including the fact that it matches closely with the P2P philosophy (present in grid systems as well) of harnessing multiple resources that in aggregate are more powerful than that of any one member. Moreover, while the distributed nature of a PIPE service could open the door to malicious users, cooperation among the "good" nodes in the system can be leveraged to deal with malicious attacks.

In particular, it is necessary to adapt existing techniques to work in a distributed, autonomous P2P architecture. For example, public-key cryptography [8] may be employed as a component of a preservation system. However, public-key techniques often require a centralized certificate authority, and this requirement must be reconciled with the P2P philosophy of "no centralized services." Other techniques that have been developed to deal with failures and maliciousness include byzantine agreement [15], distributed transaction commit [18], cryptography, replication techniques [10,4], and reputation management [14,17]. While each of these techniques may be useful, our goal here is to construct a complete framework for a PIPE system, so that we may understand the relevant challenges and then choose (or develop) appropriate techniques to meet these challenges. Moreover, it is important to ensure that the techniques used scale to potentially large networks and continue to work for potentially very long time periods. More related work is examined in Section 5.

In this paper, we discuss our vision of a PIPE system. We are not presenting any complete solutions, but rather arguing that providing the end-to-end preservation services, despite failures and maliciousness, is a hard and unsolved problem, requiring the attention of our community. To make this argument:

- We present a strawman design for a PIPE system, listing the basic services that the system provides. This design provides a framework for discussion. (Section 2)
- We discuss the main challenges presented by malicious nodes to such a system. (Section 2)
- In order to deal with these challenges, multiple techniques must be used, and we sketch how existing techniques can be used to deal with the challenges in a PIPE system. We also point out the gaps that must be filled by ongoing research. (Section 3)
- Finally, we suggest techniques that can potentially fill these gaps. (Section 4)

While it may be impossible or prohibitively expensive to *guarantee* safety from malicious nodes with any number of safeguards, our goal is to "raise the bar," making it harder for nodes to act maliciously, while preserving efficiency and scalability. We also discuss related work (Section 5) and present our conclusions (Section 6).

2 Basic PIPE architecture

2.1 Peer-to-peer architecture

The PIPE network is composed of *peers*, or archive sites, that cooperate with each other to provide a preservation service. For example, a peer may be a university library, a government agency, or a corporation. This preservation service archives and serves *digital documents*. Examples of digital documents include JPEG images, Postscript documents, audio clips, or collections of scientific measurements. Because the peers are distributed, they must communicate by sending *messages* over some underlying communications network (e.g., the Internet).

Each peer provides resources for use by other peers. For example, a peer offers storage space for copies of digital documents, processing capability to answer searches, and bandwidth for serving documents. We chose a peer-to-peer architecture for the PIPE system because it is especially well-suited for the problem of digital preservation since such a system can effectively leverage resources scattered at distributed, autonomous sites. First, the aggregate resources of the system (storage space, bandwidth, etc.) are larger than any one site has or can afford. Second, the system preserves the autonomy of sites, since each site makes local decisions when interacting with the network, rather than having to submit to a centralized controller. This makes it more likely that sites will participate in the system. Third, a peer-to-peer network is resilient to peer failures and localized network failures. This is because the network is a collection of distributed, heterogenous sites and binary communication links that operate independently, even as other nodes and links fail. Finally, because backup copies of documents are stored "on-line" at other peers, lost or corrupted copies can be quickly restored simply by retrieving a copy of the backup over the network.

2.2 PIPE services

The PIPE infrastructure should provide end-to-end preservation of digital documents. By "end-to-end preservation" we mean the system ensures that documents, once published, always exist in the system, can be discovered by users performing content-based searches, and can be retrieved by those users. More specifically, we propose a PIPE system that offers the following services:

- $join()$: Gives a new node i a list of nodes already in the PIPE network, and informs each of these existing nodes about i's existence.
- $publish(D)$: Given a document D, publishes D to the network.
- $search(q)$: Given a descriptive search q (such as keywords or metadata), returns the ids of documents that match q, and the ids of nodes holding copies of those documents.
- $retrieve(D, i)$: Given a document id D and a peer id i, retrieves a copy of the document from the peer.

In Section 3, we sketch how various techniques can be employed to implement these operations in a secure manner.

2.3 Malicious node threat model

Peers in the PIPE network cooperate to provide publishing, search and retrieval services. Unfortunately, a malicious node may appear to contribute to the service, but instead act to subvert the service. In particular, a malicious user or site may try to disrupt the PIPE service in several ways:

A. Publish a document using the same id as an existing document. Then, when a user expects one document, that user will receive the attacker's document instead.
B. Agree to store a copy of a document, but delete it instead.
C. Agree to store a copy of a document or an index of documents, but refuse to perform (some or all) *search* operations. This may prevent good nodes from finding documents.
D. Agree to serve a document, but serve an altered copy instead, or decide later not to serve the document.
E. Coordinate attacks with other malicious nodes (to avoid detection or increase the severity of the attack.)
F. Masquerade as a different peer.
G. Modify otherwise authentic messages. This disrupts the ability of good nodes to communicate.
H. Lie in response to any request for information. This disrupts the ability of good nodes to get a picture of the state of the system.

If these behaviors are surreptitious, it may be difficult to discern that the node is malicious or specifically what its malicious behavior is.

In addition to these attacks, a PIPE network is also vulnerable to malicious activities common to any distributed system, such as viruses, trojans, hackers, denial of service attacks [19], and so on. Much research is currently focused on these challenges. Techniques that are useful for general distributed systems are also useful for a PIPE network. Here, we focus on the problems unique to the PIPE network.

3 Implementing a PIPE service

In this section, we examine a "strawman" implementation of the PIPE service. This implementation must deal both with failed nodes and malicious nodes. A failed node is one that has stopped working, and no longer responds to messages. A malicious node may continue to receive, process and send messages, but may do so in order to subvert the network for its own ends. In particular, a malicious node may decide not to live up to agreements it has made with other nodes. A good, live node has neither failed nor is malicious.

3.1 Assumptions and requirements

We assume that in a network of n nodes, up to m nodes may be malicious at any one time, while up to k otherwise good nodes may have failed. The value of k might be calculated from the expected error rates in the system, while m can be chosen to represent the robustness of the system to attacks: a higher m can tolerate more malicious nodes but at a potentially high price.

We also assume that there is a bootstrap mechanism for discovering nodes that are in the network. This mechanism, which is required in all existing peer-to-peer systems, may be a centralized list of node ids, an anycast service, or some other out-of-band mechanism. This discovery mechanism does not need to maintain all of the node ids, but should maintain a significant fraction of them.

In addition to these assumption, we specify two requirements to ensure effective peer to peer communication despite malicious nodes. The first requirement is **secure, unique peer ids**. A malicious peer may try to masquerade as another peer, but this behavior should be detectable by "good" (non-malicious) nodes. Peer authentication is vital to many operations in the PIPE network. For example, it is vital that nodes accepting a copy of a document as part of a *publish* operation be able to verify that the node sending the copy is in fact the publisher and not some malicious node peddling a forgery.

There are several existing techniques that can be used to satisfy this requirement. One approach is to use an authentication authority to verify the identify of peers. However, this authority could be a single point of failure or security vulnerability. A more distributed approach is for a node to generate a public key/private key pair. The public key would be the node id, while the private key would be used to sign messages. However, if this private key needed to be matched to a real-world id (e.g., mapping the id "42355" to "Stanford University") a central authority may still be needed. One research problem is to develop techniques that provide secure, unique peer ids with minimal dependence on a central authority.

The second requirement, **secure communications channels**, ensures that good, live nodes can communicate despite failures or attacks. This component is vital as PIPE services require the cooperation of multiple nodes. By *secure* we mean that a message from peer i to peer j 1) can be authenticated as coming from i, 2) is unmodified and 3) arrives or detectably fails to arrive at j (so that i can resend). We do not require that messages be private (e.g., unreadable by third parties). Secure channels can be implemented by known, reliable transmission protocols (such as secure HTTP) although these protocols may require a public-key infrastructure. An alternative is to use dedicated

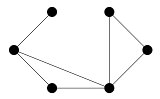

Fig. 1. A partially connected network.

physical channels, although many such channels may be necessary in a large network.

Note that it may be difficult to satisfy the requirement of secure channels if the network is *partially connected*; that is, each peer only communicates with a limited set of neighbors (Figure 1). In such a network, peers are expected to *forward* messages in addition to responding to them. A malicious node may be able to disrupt secure channels by refusing to forward messages. Therefore, for now we must make another assumption: that the network is fully connected. Research is necessary to develop techniques for secure communication channels over partially connected networks.

3.2 PIPE operations

Now we can turn to our strawman implementation of PIPE operations. Our implementation follows the basic outline of other replicated data management systems: enough copies are made of data items so that at least one copy survives failures, and the system must periodically detect and recover from failures by making new copies. However, there are certain challenges that are new in the PIPE domain. First, the replication must tolerate malicious nodes, nodes that claim to be assisting in replication but instead may interfere with the process. Also, queries locate documents by content, not by id. Content-based searching means that a replicated catalog, mapping object ids to nodes, is not sufficient. Instead a searching mechanism is needed, and must tolerate malicious nodes. In this section, we will walk through the application of replicated data management techniques to the PIPE system, and point out new challenges raised by the need to implement secure content replication and discovery. Then, in Section 4, we discuss possible new techniques to improve the efficiency of this strawman.

In a traditional replicated database system, the set of participating nodes is relatively fixed and well known to the rest of the system. However, in a PIPE network, nodes may join at any time, and it is difficult to predict which nodes will comprise the system at any instant. Therefore, we must implement the *join* operation to allow a good node to join the system securely by connecting to other good nodes even if malicious nodes are in the network.

- *join*(): Discover $k + m + 1$ nodes via the bootstrap mechanism. Contact at least $m + 1$ of them and ask for a list of nodes in the network. Contact each of the nodes in the network and announce the new node's existence.

A new node i contacts other nodes to announce its existence, but may encounter k failed nodes before finding a live node j. However, if node j is malicious, it could return a list of either malicious nodes or invalid node ids, so that i would not have a valid list of peers. If i contacts $k+m+1$ sites, it will at least get one list from a good, active node, although it may get up to m lists from malicious sites. Since i does not know which is the good list, and does not know which are invalid node ids (the nodes could simply be down temporarily), node i unions all the lists to obtain its list of nodes.

Next, we must implement *publish* to guarantee that at least one copy of each document is preserved at a good, live node, despite k failed nodes and m malicious nodes.

- *publish*(D): Store $k+m+1$ copies of document D, each copy at a different peer (including the publisher i).

Malicious nodes may delete documents, refuse to serve documents, or serve the wrong document, but there will still be a good document being served *somewhere*.

Note that even though documents are replicated, this replication process should respect any copyright restrictions on the documents. For example, access can be restricted to authorized users via digital rights management techniques or by publishing documents only to peers that have secured access. However, the PIPE service should still preserve the bits, even if copyright restricts access to them. A full treatment of copyright restrictions is both a legal and technical discussion, and is outside the scope of this paper.

When a node has a failure, we need to take steps to make more copies of data stored by the node, since that node may lose data or never rejoin the network. To do this, we implement two additional operations:

- *detect-failure*(D, i): Probe node i to ask if it is live and has a copy of D.
- *replicate*(D): Make additional copies of D until there are at least $k+m+1$ copies at live nodes.

Because the original publisher may have failed, each node that has a copy of document D should periodically perform *detect-failure* to probe all other nodes holding a copy of D. The probe could be as simple as asking for a checksum or CRC of the document. As soon as any node notices that another peer has failed or a document is lost, it can use *replicate* to ensure that there are still at least $k+m+1$ copies. *Replicate* is a lightweight operation, unlike traditional database recovery mechanisms where complex transactions must be undone or redone.

Unlike traditional systems, nodes using *replicate* must be careful about which copy is chosen as the "good" document to replicate. If an altered or incorrect version is inadvertantly chosen, then malicious behavior will be enhanced rather than mitigated. One approach is to define the document id as a secure hash of the content, for example using SHA1 or MD5. Then, a node should recalculate the signature of a document to verify its authenticity before replicating it.

A PIPE system can go beyond traditional detect and repair techniques to also mitigate the situation where a document is stored at a live but malicious node. However, detecting that a node is malicious is far harder than detecting that the node has failed. For example, if node X is behaving badly (e.g., refusing to serve documents), that is easy to detect, since *retrieve* requests are always refused. However, imagine a node Y that is waiting for the right time to strike. For most of its lifetime, it acts "good," responding to requests and serving documents. At some point Y may notice that there have been failures in the network, and that it now holds the last copy of some document D. If Y then deletes D, that document is lost forever. Such behavior is impossible to predict beforehand.

Therefore, a key research question is to determine which malicious acts can be detected. Undetectable maliciousness must be masked by extra redundancy, so that even if a node becomes unexpectedly "evil" there are enough "good" copies of documents so that the evil behavior is not detrimental. Research is also needed to balance redundancy with the resource limitations of the system.

In addition, we can mitigate maliciousness by defining operations to detect some of the more obvious malicious behaviors. For example:

- *detect-has-copy*: Choose a random portion of a document that a node is supposed to be storing, and ask the node to return that portion. If the node cannot, or the returned portion does not have the expected content, it has destroyed the document and is therefore malicious.

Each node X should take the responsibility of probing remote nodes that store copies of the same documents as X. If any detection operation indicates that a node is acting in bad faith, then extra copies of documents held by the malicious node can be made to better protect the document. The *replicate* operation can be used for this purpose. The bad node can also be ostracized from the system, although such punishment should be implemented carefully to avoid attacks based on false accusation.

We must implement the *search* operation to ensure that digital documents matching a content-based query q can be found.

- *search*(q): Broadcast the query q to all peers. Broadcast can be performed by sending an identical search message to all peers. If a peer i has a document matching the search, that peer returns the id D of the document as well as i, its own id.

Because searches are by content, multiple documents may match the query, and this implementation guarantees that results for any good, live matching documents will be returned to the user. However, in addition to "correct" search results, a malicious node may return bad results which must be filtered out by the searching node. Part of the difficulty in this process is the ambiguity in distinguishing between "junk" and "real documents." For example, imagine that a user wants to find the book "Origin of Species." There may be multiple documents titled "Origin of Species," including the classic Darwin book and also other books written by other people that are not necessarily acting maliciously. Moreover, specifying more metadata (such as the author or year) may not help. For example, a PhD thesis titled "Origin of Species by Charles Darwin in 1859" will "legitimately" match even detailed searches. In contrast, some documents are clearly "junk." If a malicious node rewrites "Origin of Species" in order to deliberately deceive readers, then the altered version is certainly junk.

An important research problem is to develop techniques for filtering out this junk. This is not an easy problem; how would a user that has never read "Origin of Species" know which document was real and which had been altered? One possibility is for some authority (such as the Library of Congress) to serve as a document authenticator. This solution unfortunately decreases the robustness of the system, since the authority becomes a single point of failure. Another potential technique is to use *secure timestamps* [16] to distinguish between documents based on creation time. While secure timestamps can be implemented in a distributed way, they may not offer enough information to properly filter junk.

Finally, after a user has peformed a *search* she might decide to *retrieve* one or more documents. We must securely implement *retrieve* to guarantee that a good, live copy of any document published to the PIPE system can be retrieved by the user.

- *retrieve*(D, i): A message containing the document id D is sent to the peer i, and i returns a copy of the document. This process is repeated until an authentic document is retrieved.

A malicious node may produce a fake or altered document in response to a *retrieve*. To avoid this, it should be possible to verify that the retrieved document matches the retrieve request. As with *replicate*, if a signature scheme is used then the user can verify the document authenticity by recalculating the signature. Since there should always be at least $k+m+1$ copies of documents, eventually the user will retrieve a good live copy.

4 Improving the PIPE operations

The techniques of Section 3.2 are "brute-force;" they rely on broadcast and a high degree of replication. A key research challenge is to develop techniques that more efficiently use resources such as bandwidth or storage while still ensuring the basic preservation properties of the system. In this section we propose starting points for developing techniques that meet the requirements of both efficiency and security.

4.1 Improving detection of malicious nodes

The *detect-has-copy* operation attempts to probe a site to see if it is operating correctly. Although this probing may detect some malicious behaviors, others may occur unnoticed. Moreover, probing requires many messages to be sent, using up network bandwidth, even when no sites are malicious. If peers are participating in the system because they want to derive benefit from it, one alternative is to use *incentive-based* verification: nodes only derive benefit from the system if they behave correctly. Incentives may not prevent all malicious actions, but can deter nodes that are merely selfish.

One example of incentive-based methods is to make nodes prove they are storing documents that they have agreed to store in order to retrieve other documents. Imagine that a node M is responsible for storing a copy of a document D_1. If M wants to *retrieve* another document D_2 from another node P, P could return $D_3 = D_1$ XOR D_2 instead of D_2. M could only read D_2 if it is properly storing D_1, and thus has an incentive not to delete D_1. This scheme requires augmenting *publish* to distribute a list of who should be storing what so that the correct test can be performed at *retrieve* time. Unfortunately, this scheme only deals with a few of the potential malicious acts, and only works if M and P store some of the same documents. More research is necessary to develop this scheme to serve as the basis of a robust incentive-based mechanism.

4.2 Improving search

Clearly, broadcast is an inefficient mechanism for implementing the *search* operation. A better alternative is to employ indexing. Indexes can be constructed over the data content or metadata, and then replicated in the PIPE network. A node performing a *search* would only have to contact a few indexing nodes to determine the location of a document or data set, rather than broadcasting its query to the whole network.

However, the use of indexing introduces the possibility that indexing nodes may be malicious. Therefore, there must be multiple copies (at least $k+m+1$) of each index, just as there are multiple copies of documents. With enough replication, malicious nodes would be unable to prevent a user from getting valid search results, although malicious nodes may still return "junk" results.

As before, junk results can be filtered by using secure timestamps or an authority like the library of Congress. With more index replication, we can discover from the indexes themselves which are the correct results. If there are $k + m \times 2 + 1$ indexes, then there will be at least $m + 1$ good, live indexes, and a peer performing a query would only accept search results that appear in at least $m + 1$ indexes. This approach has the disadvantage that more indexes must be queried for every search, but reduces the need to rely on timestamps or an authority to discover good search results.

Indexes would be updated whenever a new document was *publish*ed, or whenever a new copy was made with *replicate*. If an index is lost due to peer failure, then another indexing node must perform an *index replicate* to make a new copy of the index. It is not sufficient to simply copy an index from a live node; that node may be malicious and the index may be deliberately corrupted. As with *search*, each index entry can be verified using secure timestamps, a central authority, or by querying all indexes and accepting entries that appear at least $m + 1$ times.

After indexes are built, their location must be made known to other peers, so that those peers can conduct searches. The simplest mechanism is that a node notifies the network, using broadcast, that it has an index. This reduces the use of broadcast to a one-time setup message.

5 Related work

Replication to protect against failures has been employed in several systems, such as RAID [20], mirrored disks [3], replicated file systems [10], and so on. In most of these systems, all of the replicas are under the control of a central authority, and thus issues dealing with autonomy and maliciousness are less relevant.

A variety of techniques have been investigated for protecting against malicious activities in a distributed system. One class of techniques are based on cryptography, including secret key [1], public key [8] and cryptographic hashing [2] mechanisms. If nodes fail and lose their keys, key escrow techniques [7] can be used to recover. However, many protocols based on cryptography assume a central authority or set of authorities, for example to issue certificates. This assumption may not be appropriate in a distributed, autonomous system, especially if the authority itself may fail. Moreover, encryption only deals with some of the problems we examine (e.g., authentication) and not others (e.g., guaranteeing that a node storing a document will serve it). Another class of techniques seek to specify security in a declarative way. Systems such as PoET [21] can be used to enforce access control over digital objects, but do not address the problem of ensuring an object is stored and served.

Much recent research has focused on peer-to-peer systems. Several systems have been developed, including academic projects such as Chord [11] and Pastry [23] and industrial systems such as as Limewire and Kazaa. Much of this work has been focused on efficient and scalable search (as in Chord) or on publisher and searcher anonymity (as in FreeHaven [9]), rather than on the malicious behaviors we examine here. Peer-to-peer systems such as LOCKSS [22] and Archival Intermemory [13] deal with malicious nodes by depending on lots of extra copies, similar to what we propose in our basic techniques. Our PIPE system builds on these earlier systems by integrating search and publishing into one fault-tolerant, secure system.

6 Conclusion

Peer-to-Peer Information Preservation and Exchange (PIPE) networks are the central component of a robust, community-based digital library. In this paper we have outlined the PIPE service, including the operations it provides, the peer-to-peer architecture of cooperating nodes, and the properties (such as document preservation) it aims to achieve. We have also sketched how the system could be deployed to handle failures and malicious activities.

Examining the possible attacks, it is clear that digital libraries are quite vulnerable to malicious attacks. One could say that they are significantly more vulnerable than conventional libraries, where it is often easier to control who comes in contact with the library materials. In the digital world, any teenager with a PC can try to impersonate library servers, can inject into the system fake and altered documents, or can mount denial-of-service attacks.

We have also seen that protecting against malicious attacks is inherently costly. For every possible malicious site, we need to make extra copies of our documents. Every time we handle information, we have to be cautious: where did it come from? do several sites agree with this? when was this actually created? Even though there are potential optimizations, the bottom line is that it is still very expensive to protect against malicious users, a fact we may just have to live with.

Given the high cost of the mechanisms we have presented, it is clear that much research must be done to develop more efficient techniques. For example, it may be tempting to consider another class of "probabilistic" mechanisms. Instead of making a fixed number of copies of a document (e.g., $k+m+1$), sites would simply strive to make "lots" of copies of all documents. For example, a site may cache copies of say a random 5% of the documents it sees from other sites. The hypothesis is that, since each document has many copies, and no one knows exactly where they are, most documents will be protected. Such an approach requires additional study, but in the end we suspect that many more copies of documents will be needed, and that there will be a danger that some documents (perhaps the less popular ones) will be lost.

References

[1] FIPS PUB 197. Advanced encryption standard, 2001. *Federal Information Processing Standards Publication*, U.S. Department of Commerce, National Bureau of Standards, Springfield (Virginia).

[2] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk. Cryptographic hash functions: A survey, July 1995. Technical Report 95-09, Department of Computer Science, University of Wollongong.

[3] A. Borr. Transaction monitoring in Encompass [TM]: Reliable distributed transaction processing. In *Proc. 7th VLDB*, Sept. 1981.

[4] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *OSDI: Symposium on Operating Systems Design and Implementation*, 1999.

[5] B. Cooper, A. Crespo, and H. Garcia-Molina. Implementing a reliable digital object archive. In *Proc. European Conf. on Digital Libraries (ECDL)*, 2000.

[6] B. Cooper and H. Garcia-Molina. Creating trading networks of digital archives. In *ACM/IEEE Joint Conference on Digital Libraries, Roanoke, VA, June 2001*, pages 353–362, 2001.

[7] D.E. Denning and D.K. Branstad. A taxonomy for key escrow encryption systems. *Communications of the ACM*, 39(3):34–40, 1996.

[8] W. Diffie. The first ten years in public key cryptography. *Proc. of the IEEE*, 76(5):560–577, 1988.

[9] R. Dingledine, M.J. Freedman, and D. Molnar. The free haven project: Distributed anonymous storage service. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 67–95, 2000.

[10] B. Liskov et al. Replication in the Harp file system. In *Proc. 13th SOSP*, Oct. 1991.

[11] I. Stoica et al. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM*, 2001.

[12] John Kubiatowicz et al. OceanStore: An architecture for global-scale persistent storage. In *Proc. ASPLOS*, Nov. 2000.

[13] Y. Chen et al. A prototype implementation of archival intermemory. In *Proc. of the ACM Conf. on Digital Libraries*, 1999.

[14] S. Kamvar, M.T. Schlosser, and H. Garcia-Molina. The EigenTrust algorithm for reputation management in P2P networks. In *Proc. WWW Conference*, 2003.

[15] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. ACM Transactions on Programming Languages and Systems, Vol.4, No.3, pp. 382-401, July 1982.

[16] P. Maniatis and M. Baker. Secure history preservation through timeline entanglement. In *Proc. of the 11th USENIX Security Symposium*, Aug. 2002.

[17] Sergio Marti and Hector Garcia-Molina. Identity crisis: Anonymity vs. reputation in p2p systems. In *3rd International Conference on Peer-to-Peer Computing (P2P 2002)*, 2003.

[18] C. Mohan, R. Strong, and S. Finkelstein. Methods for distributed transaction commit and recovery using byzantine agreement within clusters of processors. In Proceedings of the Sixth ACM Symposium on Principles of Distributed Computing, 1987.

[19] D. Moore, G. Voelker, and S. Savage. Inferring internet denial of service activity. In *Proc. of 2001 USENIX Security Symposium*, August 2001.

[20] D. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). *SIGMOD Record*, 17(3):109–116, September 1988.

[21] S. Payette and C. Lagoze. Policy-carrying, policy-enforcing digital objects. In *Proc. European Conf. on Digital Libraries (ECDL)*, 2000.

[22] V. Reich and D. Rosenthal. Lockss (lots of copies keep stuff safe). Preservation 2000, Nov. 2000.

[23] A. Rowstron, P. Druschel, and P. Scalable. Distributed object location and routing for largescale peer-to-peer systems. In Proc. IFIP/ACM Middleware 2001, Heidelberg, Germany, November 2001.