POST: A secure, resilient, cooperative messaging system*

Alan Mislove[1] Ansley Post[1] Charles Reis[1] Paul Willmann[1] Peter Druschel[1]
Dan S. Wallach[1] Xavier Bonnaire[2] Pierre Sens[2] Jean-Michel Busca[2]
Luciana Arantes-Bezerra[2]

[1]Rice University, Houston, TX, USA
[2]LIP6, Université Paris VI, Paris, France

Abstract

POST is a decentralized messaging infrastructure that supports a wide range of collaborative applications, including electronic mail, instant messaging, chat, news, shared calendars and whiteboards. POST is highly resilient, secure, scalable and does not rely on dedicated servers. Instead, POST is built upon a peer-to-peer (p2p) overlay network, consisting of participants' desktop computers. POST offers three simple and general services: (i) secure, single-copy message storage, (ii) metadata based on single-writer logs, and (iii) event notification. We sketch POST's basic messaging infrastructure and show how POST can be used to construct a cooperative, secure email service called ePOST.

1 Introduction

Messaging systems like traditional email and news, as well as instant messaging, shared calendars and bulletin boards, are among the most successful and widely used distributed applications. Today, these services are implemented in the client-server model. Messages are stored on and routed through dedicated servers, each hosting a set of user accounts. This partial centralization limits availability, because a failure or attack on a server denies service to the users it supports. Also, substantial infrastructure, maintenance and administration costs are required to scale to large numbers of users. This is true in particular for semantically rich, complex messaging systems like Microsoft Exchange and Lotus Notes.

POST is a cooperative infrastructure that utilizes the untapped resources of users' desktops to provide messaging services. Unlike server-based systems, POST is *self-scaling*: the addition of new user desktops and periodic upgrades of existing desktops implicitly add more resources, thus balancing increased demands on the service due to additional users and new features. POST does not present a single point of failure or attack, and is thus potentially more resilient than server-based systems. Finally, the self-organizing properties of POST promise reduced system administration costs.

POST provides three basic services to applications: (1) persistent single-copy message storage, (2) metadata based on single-writer logs, and (3) event notification. A wide range of messaging applications can be constructed on top of POST using these services.

POST is built upon a structured p2p overlay network, providing it scalability, resilience and self-organization. Users contribute resources to the POST system (CPU, disk space, network bandwidth), and in return, they are able to utilize its services. POST assumes that participating nodes can suffer byzantine failures. Stronger failure assumptions may be unrealistic, even in scenarios where participating hosts belong to a single organization, because a single compromised node could disrupt critical messaging services or disclose confidential messages.

In this paper, we sketch the design of POST, and then describe how a cooperative, secure email system can be built using POST. Unlike conventional email services, our *ePOST* system provides secure email services by default and requires no dedicated servers. Furthermore, due to its strong sender authentication, ePOST makes efficient spam defense easier. We chose email as the initial application for POST because it is well understood, and because its high availability, reliability and security demands make it a challenging driver for POST and p2p systems in general.

The remainder of this paper is organized as follows. Section 2 provides background information on Pastry, PAST, and Scribe. Section 3 sketches the design of the POST infrastructure. In Section 4, we sketch the design of a cooperative email system as an example POST application. Section 5 discusses integrating ePOST with existing email systems. Section 6 outlines related work, and Section 7 concludes.

2 Background

POST relies on *Pastry*, a structured overlay network, as well as two basic services built upon Pastry: *PAST*, a

*This research was supported in part by Texas ATP (003604-0079-2001) and by NSF (ANI-0225660), `http://project-iris.net`.

storage system and *Scribe*, a group communication system. POST could easily be layered on similar systems like Chord/CFS, or Tapestry/OceanStore [14, 6, 9, 16].

Pastry [12] is a structured p2p overlay network designed to be self-organizing, highly scalable, and fault tolerant. In Pastry, every node and every object is assigned a unique identifier chosen from a large id space, referred to as a *nodeId* and *key*, respectively. Given a message and a key, Pastry can efficiently route the message to the node whose nodeId is numerically closest to the key.

PAST [13] is a storage system built on top of Pastry and can be viewed as a distributed hash table. Each stored item in PAST is given a 160 bit key (hereafter referred to as the *handle*), and replicas of an object are stored at the k nodes whose nodeIds are the numerically closest to the object's handle. PAST maintains this invariant regardless of node arrivals or failures. Since nodeId assignment is random, these k nodes are unlikely to suffer correlated failures. PAST relies on Pastry's secure routing [2] to ensure that k replicas are stored on the correct nodes, despite the presence of malicious nodes. Throughout this paper, we assume that at most $k-1$ nodes are faulty or unreachable in any replica set.

POST stores three types of data in PAST: *content-hash blocks*, *public-key blocks*, and *certificate blocks*. Content-hash blocks are stored using the cryptographic hash of the block's contents as the handle. Public-key blocks contain monotonically increasing timestamps, are signed with a private key, and are stored using the cryptographic hash of the corresponding public key as the handle. Certificate blocks are signed by a trusted third party and bind a public key to a name (e.g., an email address). The block is stored using the cryptographic hash of the name as the handle.

Content-hash blocks can be authenticated by obtaining a single replica and verifying that its contents match the handle. Unlike content-hash blocks, public key blocks are mutable. To prevent rollback attacks by malicious storage nodes, clients attempt to obtain all k replicas and choose the authentic block with the most recent timestamp. Certificate blocks require a signature verification using the public key of a trusted third party.

Scribe [3] is a scalable multicast system built on top of Pastry. Each Scribe group has a 160 bit *groupId*, which serves as the address of the group. The nodes subscribed to each group form a multicast tree, consisting of the union of Pastry routes from all group members to the node with nodeId numerically closest to the groupId.

3 POST Architecture

POST provides three basic services: a shared, secure single-copy message store, metadata based on single-writer logs, and event notification. These services can be combined to implement a variety of collaborative applications, like email, news, instant messaging, shared calendars and whiteboards.

A typical pattern is that users create messages and insert them in encrypted form into the secure store. To send a message to another user or group, the notification service is used to provide the recipient(s) with the necessary information to locate and decrypt the message. The recipients may then modify their personal metadata to incorporate the message into their view (e.g., into a private mail folder).

POST assumes the existence of a certificate authority. This authority signs certificates binding a user's unique name (e.g., her email address) to her public key. The same authority issues the nodeId certificates required for secure routing in Pastry [2]. Furthermore, the authority may set policies for each user (such as ensuring that each user owns a nodeId bound to a live IP address), thus forcing the user to contribute resources to the system. Users can access the system from any node, but it is assumed that the user trusts her local node, hereafter referred to as the trusted node, with her private key.

Throughout the design of POST, we assume that objects stored in PAST cannot be deleted. Thus, the amount of available disk space in the system must be increasing and greater than the total storage requirements, which is reasonable to expect in a p2p environment where each participant is required to contribute a portion of her desktop's local disk.

3.1 User Accounts

Each user in the POST system possesses an account, which is associated with an identity certificate. The certificate is stored as a certificate block, using the secure hash of the user's name as the handle. Also associated with each account is a user identity block, which contains a description of the user, the contact address of the user's current trusted node, and any references to public metadata associated with the account. The identity block is stored as a public-key block, signed with the user's private key. Finally, each account has an associated Scribe group used for notification, with a groupId equal to the cryptographic hash of the user's public key.

The immutable identity certificate, combined with the mutable public-key block, provides a secure means for a trusted authority to bind names to keys, while giving users the ability to change their personal contact data without requiring subsequent interactions with the certificate authority. The Scribe group allows anybody waiting for news from that user, or anybody wishing to notify the user that new data is available, to have a common rendezvous point.

3.2 Secure Message Storage

POST provides a shared, secure message storage facility. Application-provided message data is encrypted using a technique known as convergent encryption [7]. Convergent encryption allows a message to be disclosed to selected recipients, while ensuring that copies of a given cleartext message inserted by different users map to the

same ciphertext, thus requiring only a single copy of the ciphertext to be stored.

When an application wishes to store message X, POST first computes the cryptographic $Hash(X)$, uses this hash as a key to encrypt X with an symmetric cipher, and then stores the resulting ciphertext at the handle

$$Hash\left(Encrypt_{Hash(X)}(X)\right)$$

which is the secure hash of the ciphertext. To decrypt the message, a user must know the hash of the plaintext.

Convergent encryption reduces the storage requirements when multiple copies of the same content tend to be inserted into the store independently. This happens commonly in cooperative applications, for instance, when a given popular document is sent as an email attachment or posted on bulletin boards by different users.

Convergent encryption is vulnerable to certain known plaintext attacks. An attacker who is able to guess the plaintext of a message can verify its existence in the store, but cannot necessarily find out who inserted it. Moreover, since users normally encrypt their private metadata, it is not possible to determine who references the message. Nevertheless, with sensitive content, convergent encryption must be used with care, particularly when the content is of a small size, is highly structured, or is otherwise predictable. In such cases, convergent encryption could be supplemented or replaced by conventional cryptographic methods. A simple change could be to prepend some number of random bits to the plaintext prior to the convergent encryption.

3.2.1 Scoped storage overlays

P2p storage systems like PAST or CFS form a single overlay network that includes all participants. Replicas of stored objects are placed at random nodes with adjacent nodeIds throughout this overlay. This approach leads to good load balancing and failure independence, since the set of replica nodes for an object is widely distributed and thus unlikely to suffer correlated failures.

On the other hand, network locality can be poor because all objects are replicated at global scope, even when an object is only of local interest and a more local distribution (e.g., within a large organization) may yield adequate failure independence. The lack of centralized node administration makes it difficult to assess individual nodes' failure probabilities, and thus determine the appropriate degree of replication. And, the fact that any node can insert objects anywhere in the system invites denial-of-service attacks aimed at exhausting the storage space of certain nodes, or the entire system. Lastly, it is difficult to let nodes behind a firewall participate in the storage overlay.

POST overcomes these problems using a two-level store consisting of organizational overlays and a global overlay. The two-level store allows POST to scope the insertion of documents into the store, such that documents inserted by members of an organization are replicated among the organization's nodes. This is achieved without sacrificing load balancing, failure independence, or the ability to look up a stored message anywhere in the global overlay; we omit the details due to lack of space.

3.3 Event notification

The event notification service is used to alert users to certain events, such as the availability of a message, a change in the state of a user, or a change in the state of a shared object.

For instance, after a new message was inserted into POST as part of an email or news service, the intended receiver(s) must be alerted to the availability of the message and provided with the appropriate decryption key. Commonly, this type of notification requires obtaining the contact address from the recipient's identity block. (This may require a lookup of the recipient's certificate block, if the certificate is not already cached by the sender). Then, a notification message is sent to the recipient's contact address, containing the secure hash of the message's ciphertext and its decryption key, encrypted with the recipient's public key and signed by the sender.

In practice, notification can be more complicated if the sender and the recipient are not on-line at the same time. To handle this case, the sender may delegate the responsibility of delivering the notification message to a set of k random nodes; we omit the details here due to lack of space.

To guarantee confidentiality, each notification message is encrypted using a symmetric cipher such as AES with a unique session key, and the session key itself is then encrypted using the recipient's public key. Thus, only the recipient can decrypt the session key (i.e., with his private key) in order to decrypt the remainder of the message. Each notification message is also signed with the sender's private key, allowing the recipient to verify its authenticity. Finally, each notification message also includes a timestamp to prevent the message from being replayed by malicious users. Note that, unlike most traditional user messaging infrastructures, everything in POST is digitally signed and encrypted, by default. This will prove useful when implementing higher-level services like email, chat, and so forth.

3.4 Metadata

POST provides single-writer logs that allow applications to maintain metadata. Typically, a log encodes a view of a specific user or group of users and refers to stored messages. For instance, a log may represent updates to a user's private email folder, or a public news group. An email or news application would use a log of insert and delete records to keep track of the state of a user's mail folder or a shared folder representing a news group.

In general, logs can be used to track the state of a chatroom, a newsgroup, a shared calendar, or an arbitrary data structure. POST represents logs using self-authenticating blocks that form a content-hash chain. This is similar to, and was inspired by, the logs used in the Ivy p2p filesystem [11].

The log head is stored as a public-key block and contains the location of the most recent log record. Handles for log heads may be stored in the user's identity block, in a log record, or in a message. Each log record is stored as a content-hash block and contains application-specific metadata and the handle of the next recent record in the log. Applications optionally encrypt the contents of log records depending on the intended set of readers.

In the original implementation used in Ivy, the log head and each log record are stored at a different set of nodes. To allow for more efficient log traversal, POST stores clusters of M consecutive log records on the same node, under the handle of the least recent of the M records. To deal with partially filled clusters, the log head contains an additional handle, referring to the least recent record in a partially filled cluster. This handle identifies the cluster.

Other optimizations are possible to reduce the overhead of log traversals, including caching of log records at clients and the use of snapshots. Like Ivy, POST applications may periodically insert snapshots of their metadata into the store. Thus, log traversals always terminate at the most recent snapshot.

3.5 POST robustness and security

The single-writer property and the content-hash chaining [10] of the logs make it very hard for a malicious user or storage node to insert a new log record or to modify an existing log record without the change being detected. To prevent version rollback attacks, public-key blocks contain version timestamps. When reading a public-key block (e.g., a loghead) from the store, clients attempt to read all k replicas of the block, and use the authentic replica with the most recent timestamp. When reading content-hash blocks or certificate blocks, it is sufficient to use any authentic replica.

Of great concern is the durability of stored messages. It depends on the failure independence of the replica node sets and an appropriate choice of replication factor, relative to the failure rate of individual nodes. POST's scoped insertion into local overlays greatly eases the assessment of failure independence and node failure rates, because all nodes are under some level of joint administrative control.

Organizations that run a local overlay should ensure that nodes are spread over different buildings, if not different sites. To reduce the risk of correlated failures due to security attacks, there should be sufficient heterogeneity in hardware and software. This can be difficult to ensure due to most organizations' monoculture approach to systems administration. However, risks from common virus attacks can be greatly reduced by running the POST daemon with reduced system privileges under its own user identifier. Thus, a compromised POST daemon has insufficient privileges to cause harm to the rest of the system. Likewise, other compromised user applications cannot attack POST's local file store.

Pastry's secure routing mechanism provides an effective defense against denial-of-service attacks against the overlay, both from within and outside [2]. Attacks aimed at filling the store can be thwarted with relative ease due to the use of local overlays. Since object insertions are allowed only within a local overlay, it is possible to track, identify and reprimand offenders within an organization.

Single-writer logs are the only mechanism used to maintain mutable state in POST. Their use avoids the cost and complexity of a general byzantine fault-tolerant replicated state machine. We are confident that POST's restricted mechanism for mutable state is flexible enough for applications like email, news, instant messaging and calendaring. The logs are efficient in cooperative applications, where insertions occur at a rate typical of human user actions.

Some cooperative applications may require a more flexible mechanism for maintaining mutable state. To support such applications, the authors at LIP6 are currently investigating additional, byzantine fault-tolerant mechanisms for maintaining multi-writer, mutable state.

4 Example: Electronic mail

In this section, we sketch the design of a serverless email system, ePOST, on top of the POST infrastructure. The goal is to show how POST can support a secure, scalable and highly resilient email system that leverages the resources of participating desktop computers.

While a system like ePOST promises increased resilience, greater scalability and lower cost, it remains an open question whether these advantages will be sufficient to completely displace the existing, server-based email infrastructure. Nevertheless, we chose to pursue ePOST for several reasons.

First, ePOST is designed so that it can be deployed incrementally, thus allowing individual organizations to adopt it while still relying on existing standards and infrastructure for communication across organizations. Second, unlike most existing p2p applications, email is mission-critical and demands high reliability, security, and availability. Thus, it is a challenging driver for the development of POST and, more generally, the underlying p2p infrastructure.

4.1 Overview

Each ePOST user is expected to run a daemon program on his desktop computer that implements the Pastry, PAST, Scribe and POST protocols, and contributes some CPU, network bandwidth and disk storage to the system. The daemon acts as a SMTP and IMAP server, thus allowing the user to utilize conventional email client programs. The daemon is assumed to be trusted by the user and holds the user's private key. No other nodes in the system are assumed to be trusted by the user (other than the authority that signs the users' certificates).

4.2 Email storage

In ePOST, email messages received from an email client program are parsed and the MIME components of the message (message body and any attachments) are stored as separate messages in POST. Thus, frequently circulated attachments are stored in the system only once.

The message components are first inserted into POST by the sender's ePOST daemon; then, a notification message is sent to the recipient. Sending a message or attachment to a large number of recipients requires very little additional storage overhead beyond sending to a single recipient. If messages are forwarded or sent by different users, the original message data does not need to be stored again; the original message reference is reused.

Due to the necessary data replication in PAST, the storage overhead per message is higher in POST compared to a conventional server-based email system. However, this effect is partly offset by POST's single-copy store, which eliminates large amounts of duplication due to large, widely circulated email attachments. Moreover, exploiting the typically underutilized disk space on desktop computers should more than compensate for this overhead [1]. Lastly, the storage requirements can be further reduced by using erasure codes [15], but we have not yet explored their use in POST.

4.3 Email delivery

The delivery of new email is accomplished using POST's notification service. A sender first constructs a notification message containing basic header information, such as the names of the sender and recipients, the subject, a timestamp, and a reference to the body and attachments of the message. The sender then requests the POST service to deliver this notification to each of the recipients.

It is noteworthy that ePOST extends recipient control beyond current systems by allowing the recipient to append the message to his mailbox or to simply ignore the notification, perhaps based on a spam filter. Since messages are stored in the sender organization's ring, one of the major goals of anti-spam researchers, to push most of the costs of spam back onto the spammers, can be achieved in a straightforward manner.

4.4 Email folders

Each mail folder is represented by a POST log. Each log entry represents a change to the state of the associated folder, such as the addition or deletion of a message. Furthermore, since the log can only be written by its owner and its contents are encrypted, ePOST preserves the expected privacy and integrity semantics of current email systems with storage on trusted servers.

An email insertion record contains the content of the message's MIME header, the message's handle and its decryption key, and a signature of all this information, taken from the sender's original notification message. This data is then encrypted under the user's public key.

4.5 Discussion

By default, ePOST provides strong confidentiality, authentication and message integrity. The system is able to tolerate up to $k-1$ faulty or unreachable nodes in any random set of k POST nodes without loss of data or service, where k is the degree of message replication. It relies on Pastry's secure routing facilities [2], data replication, and cryptographic techniques to achieve robustness under a wide range of attacks, including denial-of-service and participants that suffer byzantine faults.

More analysis and experimentation will be necessary to determine appropriate assumptions about the fraction of faulty nodes in various environments, and appropriate levels of replication. Results of a prior study on p2p filesystems in corporate environments indicate that modest levels of replication can yield high availability [1].

Since ePOST inserts all incoming messages into the local overlay, only the node failure probability and failure independence within a user's local overlay determine the durability of the messages that the user references. Therefore, a user's organization can take appropriate steps to ensure failure independence and determine an appropriate degree of replication.

Mailing lists can be easily supported by maintaining the list as an additional log and storing the log head reference at the list maintainer's user identity block. When delivering a message, the sender notices the list and expands the recipient list appropriately.

5 Incremental deployment

To allow an organization to adopt ePOST as its email infrastructure, ePOST must be able to interoperate with the existing email infrastructure. We sketch here how ePOST could be deployed in a single organization and interoperate with email services in the general Internet.

For inbound email, the organization's DNS server provides MX records referring to one of a set of POST nodes within the local organization. These nodes act as incoming SMTP mail gateways, accepting messages, inserting them into POST, and notifying the recipient's nodes. Suitable headers are generated such that the receiver is aware the message may have been transmitted on the Internet unencrypted. If no identity block can be found for the recipient in the local overlay, then the email "bounces" as in server-based systems.

The incoming mail gateway nodes need to be trusted to the extent that they receive plaintext email messages for local users. Typically, the desktop workstations of an organization's system administrators can be used for this purpose. These administrators own root passwords that allow them to access incoming email in conventional, server-based systems. Thus, ePOST provides the same privacy for incoming email from non-ePOST senders as existing systems.

Sending email to the outside world first requires determining that the desired email address is not already available inside the ePOST world. At that point, there

may be a gateway service that can provide the appropriate certificate material to generate a standard cryptographic email in S/MIME or PGP format. This encryption is performed in the sending user's local node, before the data goes onto the network. If the recipient does not support secure email, then the email must ultimately be transmitted in the clear, so the ePOST proxy server can speak regular SMTP to the recipient's mail server.

6 Related work

Lotus Notes and Microsoft Exchange provide a general, secure messaging infrastructure based on the client-server model, providing the ability to transfer email, personal contacts, calendars, and tasks. POST aims to provide similar functionality based on a serverless, decentralized and cooperative p2p architecture.

Current email protocols, including SMTP, POP3, and IMAP, are tailored towards an infrastructure based on dedicated servers. Minimal security is provided in these protocols, as they do not provide confidentiality, verifiability, or data integrity. Extensions like PGP provide secure email, but are not widely used.

The use of a single-writer, self-authenticating log in POST was inspired by the use of similar logs in the Ivy filesystem [11]. The loghead is the root of a Merkle hash tree [10], which allows the log to be stored on untrusted nodes, while ensuring that the authenticity of each log entry can be verified locally. This allows POST to avoid more complex byzantine state machine protocols [4].

A serverless email system proposed in [8] shares many of the goals of ePOST. Unlike POST, it focuses on email service only, and unlike ePOST, it is not compatible with the existing email infrastructure. Providing email services on top of a p2p storage system has also been explored in the OceanStore project [5]. The use of single-writer logs allows POST to achieve similar functionality with significantly less complexity, while providing general support for collaborative applications.

7 Status and conclusions

POST is a decentralized, collaborative messaging system that leverages the resources of participating desktop computers. POST provides highly resilient and scalable messaging services, while ensuring confidentiality, data integrity, and authentication. The basic services provided by POST can be used to support a variety of collaborative applications. In this paper, we have sketched how POST can be used to construct ePOST, a cooperative, secure email system.

Prototype implementations of POST and ePOST exist and are currently under experimental evaluation. Implementations of calendaring and instant messaging applications are underway. We plan to start using ePOST shortly, initially within our research groups, and hope to expand the user base within Rice and LIP6 and beyond, as we gain experience and confidence in the system. Given users' dependence on email services, we view this as a proof of concept for mission-critical p2p systems, and as a vehicle to gain practical experience and workload trace data from such a system.

References

[1] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop PCs. In *Proc. SIGMETRICS'2000*, Santa Clara, CA, 2000.

[2] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Security for structured peer-to-peer overlay networks. In *Proc. of the Fifth Symposium on Operating System Design and Implementation (OSDI 2002)*, Boston, MA, December 2002.

[3] M. Castro, P. Druschel, A-M. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC*, 20(8), October 2002.

[4] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *OSDI: Symposium on Operating Systems Design and Implementation*. USENIX Association, Co-sponsored by IEEE TCOS and ACM SIGOPS, 1999.

[5] S. Czerwinski, A. Joseph, and J. Kubiatowicz. Designing a global email repository using OceanStore, June 2002. UC Berkeley summer retreat, `http://roc.cs.berkeley.edu/retreats/summer_02/slides/czerwin.pdf`.

[6] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proc. ACM SOSP'01*, Banff, Canada, October 2001.

[7] J. Douceur, A. Adya, W. Bolosky, D. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Proc. of the International Conference on Distributed Computing Systems (ICDCS 2002)*, Vienna, Austria, July 2002.

[8] J. Kangasharju, K. Ross, D. Turner, J. Syrjala, and D.S. Digeon. Peer-to-peer e-mail, November 2002. Submitted for publication.

[9] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent store. In *Proc. ASPLOS'2000*, Cambridge, MA, November 2000.

[10] R. Merkle. A digital signature based on a conventional encryption function. In *In Advances in Cryptology—CRYPTO'87 (LNCS, vol. 293)*, 1987.

[11] A. Muthitacharoen, R. Morris, T. Gil, and B. Chen. Ivy: A read/write peer-to-peer file system. In *Proc. of the Fifth Symposium on Operating System Design and Implementation (OSDI 2002)*, Boston, MA, December 2002.

[12] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Middleware 2001*, Heidelberg, Germany, November 2001.

[13] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proc. ACM SOSP'01*, Banff, Canada, October 2001.

[14] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. ACM SIGCOMM'01*, San Diego, CA, August 2001.

[15] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Proc. ITPTS'02*, Cambridge, MA, March 2002.

[16] B.Y. Zhao, J.D. Kubiatowicz, and A.D. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB//CSD-01-1141, U. C. Berkeley, April 2001.