2008 11th IEEE International Conference on Computational Science and Engineering

An experimental peer-to-peer e-mail system

Edson Kageyama, Carlos Maziero, and Altair Santin
Graduate Program in Computer Science
Pontifical Catholic University of Paraná, Brazil
{edson,maziero,santin}@ppgia.pucpr.br

Abstract

Conventional e-mail systems are prone to problems that affect their dependability. E-mail systems operate following a "push-based" approach: the sender side server pushes the e-mails it wants to send to the corresponding receivers' servers. This approach may impose processing and storage overhead on the receiver side. This paper presents a peer-to-peer e-mail system in which messages are sent directly from senders to receivers using a "pull-based" approach. The sender stores locally all e-mails it intends to send, and notify their receivers using a global, distributed notification service. Receivers can then retrieve such notifications and decide if they want to receive the corresponding e-mails. If so, e-mails can be retrieved directly from their senders. This proposal is inspired from file sharing peer-to-peer systems, in which users locate and retrieve the contents they are looking for. A prototype was built to show the feasibility of the proposal, and experimental results show its viability.

1 Introduction

Internet e-mail has become an extremely pervasive communication tool, due to its low cost, asynchronous operation and ability to easily transport different types of digital content. However, protocols for sending and receiving e-mails were designed to be simple, mainly because their initial use was restricted to the academic community and the e-mail servers' processing capacity was limited. With the wide-spreading use of e-mail, some users started using this system to propagate malicious content, as *virus*, *spams*, and *scams*. Moreover, the increase in the use of e-mail systems led to the necessity of technologies capable to support this service in a more secure, reliable, and scalable manner.

Traditional e-mail systems operate following a "push-based" approach: the sender's e-mail server pushes the e-mails she wants to send to the corresponding receivers' servers. The receiver server should then accept the e-mail and deliver it to its destination mailbox. The receiver server can implement mechanisms to avoid receiving spam, like e-mail content analysis [15] and server white lists [5]. However, most techniques impose processing and storage overhead on the receiver side.

This article presents a distributed e-mail architecture in which e-mail is transported according to a "pull-based" approach, directly from senders to receivers. The sender stores locally all e-mail it intends to send, and notifies their receivers using a notification service built using a distributed hash table [13]. Receivers can then retrieve such notifications and decide if they want to receive the corresponding e-mails. If so, e-mails can be retrieved directly from their senders.

This article is structured as follows: section 2 reviews the conventional e-mail structure and the main threats to e-mail services; section 3 reviews some related work on *peer-to-peer* based e-mail systems; section 4 presents the architecture developed in this project; section 5 presents some prototype implementation details and discusses experimental results obtained from it; finally, section 6 concludes the paper, discussing open problems and outlining perspectives.

2 The e-mail service

The Internet e-mail architecture [7] contains agents acting as senders and receivers of e-mails. The *Mail User Agent* (MUA) is the program used by the end user for reading and writing e-mails. The *Mail Transport Agent* (MTA) is responsible for receiving e-mails from the MUAs, forwarding them to their destinations (other MTAs), receiving e-mail from other MTAs, and keeping mailboxes for local users. This simple and straightforward structure was responsible for the e-mail popularity. On the Internet, e-mail systems gained importance and visibility, showing limitations and fragilities not initially considered, like privacy (e-mails are transferred in clear text format), sender authentication (SMTP does not provide effective mechanisms to authenticate senders, allowing e-mail forgery), and efficiency (e-mails can carry large attachments, but are sent and stored uncompressed).

978-0-7695-3193-9/08 $25.00 © 2008 IEEE
DOI 10.1109/CSE.2008.9

Due to such fragilities, problems like spam, scam, and virus propagation compromise the security, performance, robustness, and usability of the current e-mail systems. Several techniques can be used to improve current e-mail systems, like *white/black lists* [5], *content filtering* [15], and *sender authentication* mechanisms [1, 8, 17, 19].

3 Peer-to-peer and e-mail

Peer-to-peer (*p2p*) systems present some advantages over traditional client/server systems, like more flexibility and scalability, and smaller costs. Primary *p2p* use was in anonymous file sharing over the Internet, but other application domains are appearing, like audio/video streaming, instant messaging, and distributed storage.

In the *p2p* approach, participants (*peers*) acts as clients and servers for the service being provided. Peers interact among each other to announce resources, to locate resources, and to retrieve/use them. Peer-to-peer systems can use a central server to maintain resource location and authentication services; others are decentralized and define "super-peers" to carry out resource location algorithms. Some are unstructured topology, while others provide a predefined topology to ease resource location [2].

A *Distributed Hash Table* (DHT) [13], is a decentralized *p2p* service that stores [*key, value*] pairs. This service is generally provided by a large amount of peers, allowing to store and retrieve information reliably and efficiently. Some DHT systems associate a password p and a validity time t to each entry: p protects the entry against unauthorized removals, and t allows the DHT to purge old entries. DHTs are used to build more complex services, like resource lookup, distributed file systems, and naming systems. Typical DHT implementations include *Chord* [16] and *Pastry* [14].

Studies were done on using the *p2p* approach to improve e-mail systems. In [6], the authors use a DHT for storing user certificates, e-mails and even mailboxes. For sending an e-mail, an *User Agent* (UA) retrieves the receiver certificate from the DHT, encrypts the e-mail body using the receiver's public key and stores it back on the DHT. It also stores the encrypted message headers in the DHT, using the receiver's e-mail address as the key. On the other side, the receiver UA uses its e-mail address to lookup for new message headers in the DHT; after decrypting the header, the UA uses the message ID as a key to retrieve the corresponding message body.

The *ePOST* system[10] builds an e-mail system on top of a *p2p* storage infrastructure called *POST* [9], which is built on top of the *Pastry* DHT [14]. In *ePOST*, each peer contributes to the DHT service and to the distributed e-mail storage. E-mails and metadata are encrypted and stored in the DHT. The local agent acts as a SMTP/IMAP server, allowing the use of conventional e-mail clients. E-mail header, body, and attachments are stored separately; thus, a file sent as attachment to several destinations is stored only once in the DHT. E-mail delivery is done through the *Scribe* notification service: the sender posts a notification to each receiver, containing the message header and references to the message parts.

The work shown in [18] uses an hybrid *p2p* model, composed by communities and a central server, providing authentication and peer location services. Each community comprises a set of nodes and a super-node. The super-node acts as a conventional MTA: all messages in a community are sent to its super-node, which delivers them to nodes in its community or forward them to other super-nodes. Alternatively, super-nodes can route the sending requests between peers, letting the e-mail transfers occur directly between them. If a super-node fails, its community elects another super-node. Finally, *broker nodes* can send messages to conventional e-mail servers.

To our knowledge, all systems proposed rely on *pushing* e-mails to a local server (super-node or local SMTP server), to a remote storage (DHT or remote SMTP server), or directly to the destination nodes. Consequently, the storage overhead does not affect the sender itself, but others. Also, spam/virus filtering should be done by the receivers, *after* receiving the e-mail. Finally, current DHT implementations have small limits on the size of the information that can be stored under each key, preventing the transfer of large messages. The next section proposes an architecture in which e-mails are stored in the sender side until they are *pulled* out by their destination nodes, if they accept to receive them.

4 A pull-based p2p e-mail system

This article presents a *p2p* e-mail architecture in which messages are transported according to a "pull-based" approach, directly from senders to receivers. The system is composed by a set of peers, each one acting for an user. Concerning the message flow, the system follows an unstructured approach, in which all peers have the same features. The e-mails are exchanged directly between the peers, there are no central authentication or storage servers. A DHT service is used to forward notifications from senders to receivers, and to store some control information.

A peer is a daemon that acts as an SMTP/IMAP server for its local client. Message format respects the RFC2822 specifications [12], allowing users to use standard client software (MUA) for e-mail handling. Each peer *p2pMTA* has four storage areas: a *spool area*, to store messages sent by the local MUA; an *inbox area*, in which it deposes messages to be read by the local MUA; a publicly accessible *outbox area*, holding messages to be pulled by receivers using HTTP; and a *key cache area*, to store public PGP keys of known peers. PGP keys [19] are used for encrypting

messages and notifications, to guarantee their integrity and privacy. Peers organize themselves in groups, to improve message availability (see section 4.5).

Figure 1 shows an overview of this architecture; next sections detail how messages are transferred, and the mechanisms used for ensuring message privacy, integrity, and availability.

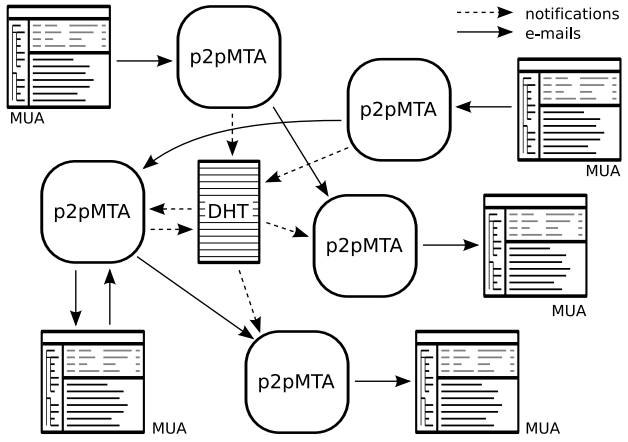

Figure 1. Architecture overview

4.1 Assumptions

We assume that the DHT provides three basic services:

- *put(x,v,p,t)*: register an entry defined by a key x, a value v, a password p, and a validity time t; the DHT supports key collisions: two or more values can be stored under the same key x;
- *get(x)*: retrieves all entries $e_i = [x_i, v_i]$ in which $x_i = x$;
- *del(x,v,p)*: deletes an entry defined by $[x, v]$, protected by a password p.

It should be noticed that password p only protects the entry against *del* operations, but does not forbid *get* operations. We consider that the DHT infrastructure purges all entries whose validity time was expired, according to [13].

Peer e-mail addresses have the format *name@group*, where *name* define the peer identity and *group* is the group of peers to which it belongs. We assume that each peer has a pair of public/private PGP keys; its public key k is available to the other peers, while its private key k' is securely kept by the peer. All hash operations are done using SHA1.

4.2 Starting an e-mail peer

When it starts, each peer i posts a *peer descriptor* d_i in the DHT, using its e-mail address as key. The descriptor d_i value contains the IP/port of the peer's outbox. The password p is chosen by the peer, and the validity time t is arbitrarily set as one day. This descriptor should be kept in the DHT by the peer, which renews or updates it when needed. If the peer leaves the network without removing its descriptor, the validity time t will ensure that it will be eventually purged from the DHT.

4.3 Sending and receiving an e-mail

The steps needed to transfer an e-mail m_i from a sender peer S to a receiver peer R are explained in the following; they are also presented in figure 2:

1. m_i is sent by the client MUA to its local peer S using SMTP; the message is stored in the spool area of S;
2. S retrieves k_r from its key cache, or from R, using the URL `http://`*ip*`:`*port*`/pubkey` (*ip* and *port* are obtained from d_r). If no key is found, an error is raised;
3. m_i is encrypted using k_r: $m'_i = k_r\{m_i\}$, and a SHA1 hash h_i is generated from m'_i: $h_i = SHA_1(m'_i)$;
4. m'_i is then stored in S's *outbox*, using h_i as its file name;
5. S posts a send notification ns_i to R in the DHT, using *ns:peer@group* as key (where *peer@group* is R's address). The ns_i value contains some m_i headers (*sender@group*, *subject*, *date*, *size*, name/type/size of attachments), h_i, and the DHT entry password p_i (a random value); ns_i value is encrypted using k_r. The validity time t is arbitrarily set to one week. ns_i is also copied locally in S.
6. Periodically, R searches for ns entries in the DHT; each ns_i is decrypted using its private key k'_r, to retrieve the message headers, the hash h_i, and the password p_i.
7. For each ns_i, R creates a message n_i in the local inbox area, using the m_i headers and an empty body;
8. The IMAP client at R retrieves all inbox messages and presents them to the user;
9. If the user decides to open n_i, R retrieves m'_i using the URL `http://`*ip*`:`*port*`/outbox/`h_i (*ip* and *port* are obtained from d_s); m'_i is decrypted using k'_r to obtain m_i. Otherwise, if the user deletes an e-mail without reading it, the corresponding inbox entry is deleted.
10. After m'_i being retrieved or n_i deleted, ns_i is deleted from the DHT, and a receive notification nr_i is posted, using *nr:peer@group* as key (where *peer@group* is

S's address). nr_i value is the hash h_i, and the same p_i from ns_i is used to protect the DHT entry.

11. Periodically, S searches for nr entries for its pending messages; for each nr_i, it removes the DHT entry and the corresponding files from its outbox.

In this approach the storage overhead remains at the sender side. A spammer trying to send thousands of messages will consume its own storage space until the receivers decide to get them, to delete them, or to ignore them. Another positive aspect of this approach: a message already sent by the user (MUA) can be canceled, if the receiver did not yet receive it.

4.4 Multiple receivers

The steps shown in the last section present the system behavior in a simple case, in which a peer send a message to a single receiver. Now, a multiple-receiver message transfer scenario is depicted. This scenario is frequent in mailing lists, for instance. The changes in the previous scenario when sending a message m_i to several receivers R_x are:

- m_i should be encrypted with the public key k_x of each receiver R_x. PGP does this easily: it cyphers m_i using a random session key ks ($m'_i = ks\{m_i\}$), then cyphers ks using k_x of each R_x, to get cyphered keys kc ($kc_x = k_x\{ks\} \forall R_x$); all kc_x are then appended to m'_i;
- Next, S should post a ns_x for each receiver R_x;
- Only when all ns_x are replied with nr_x or are expired, m_i files stored in S can be removed.

It is easy to see that the message is stored only once, even if the message has several receivers.

4.5 Peer groups

In a conventional e-mail system, the mail server acts as a temporary storage for e-mails, if their destination servers are offline. Our architecture proposal has no e-mail servers or intermediate temporary storage areas. This could cause problems, because sender and receiver should be both online to allow an e-mail to be transferred. To circumvent this restriction, peers are organized in *peer groups*.

Peers in a group replicate the contents of their outboxes, to provide higher availability of the e-mails awaiting to be retrieved by their receivers. A peer group is formed by peers that have some trust in each other. Each group has a name *gname* and a descriptor gd_i registered in the DHT under a *group:gname* key. The descriptor value contains a list of the group members' addresses. For each group, a special peer called *group master* is responsible for registering and maintaining its group descriptor. Currently, the group master is manually defined, as well as group members; more sophisticated techniques for building and maintaining groups are being investigated.

The outbox replication procedure is fairly simple: a given peer in a group retrieves lists of messages contained in each other peers' outboxes, from their URLs `http://`*`ip`*`:`*`port`*`/outbox/msglist`. Then it compares the list contents with the e-mails present in its local outbox; e-mails no more in the list should be deleted, because they were deleted by their owner; e-mails not in the outbox should be retrieved, to be replicated.

During a message transfer, if the sender peer S is offline, the receiver peer R can retrieve the sender group descriptor, to discover other peers where the message m'_i may be replicated. It will then try to retrieve the message from one of them (step 9 in section 4.3).

4.6 Garbage Collection

There are a fair amount of information being stored locally (at each peer) and in the DHT. This information should be deleted when no more needed:

- All DHT entries have validity times associated to them; when the entry age surpasses its validity time, the DHT infrastructure removes it, transparently [13].
- E-mails stored in the sender's outbox are deleted when (a) the receiver retrieved it and posted a corresponding receive notification; or (b) when the validity time of the corresponding entry in the DHT finishes.
- E-mails replicated among the group members are deleted during the replication procedure, if their owners deleted them from their respective outboxes.

5 Prototype

To show the feasibility of this proposal, a prototype was built using *Perl* and COTS components: the *OpenDHT* infrastructure [13] (a deployment of the *Bamboo* DHT implementation on the *PlanetLab* distributed platform [11]); the *GnuPG* package, an open source implementation of the *OpenPGP* standard; *Lighttpd*, a lightweight open source HTTP server; the *JES - Java E-mail* SMTP server; and the *Mozilla Thunderbird* e-mail client. The prototype provides most services defined in the previous sections, including message replication.

Some experiments were carried out to evaluate the performance of this approach. Tests consisted on sending 1000 messages to a single receiver, using messages with an empty body and a file attachment. Two types and four sizes of files

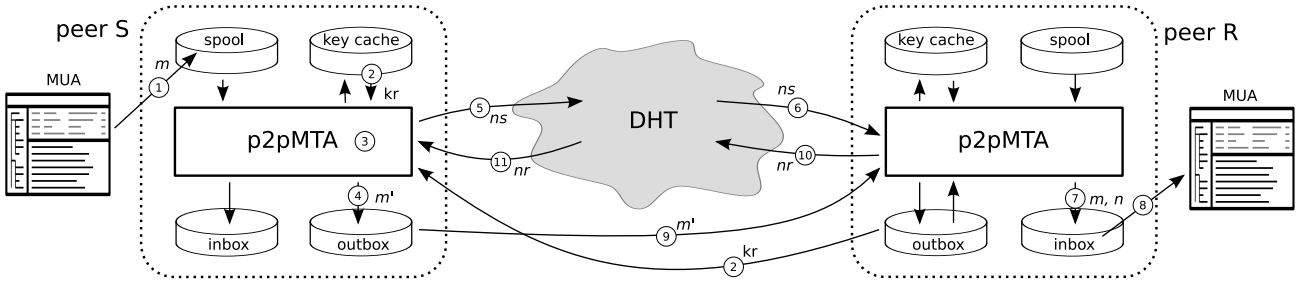

Figure 2. Sending/receiving an e-mail

were used: plain text or binary data, with sizes of 1K, 10K, 100K, or 1Mbyte. All tests were done in a local network, using Pentium IV 2.0 GHz computers to run the *p2pMTA* agents. As the latency times observed in the *OpenDHT* infrastructure were high and unstable, a set of local *Bamboo* servers was used to provide the DHT service. The same tests were repeated on a conventional SMTP system, for comparison.

Figure 3 shows the time needed to receive 1000 messages in *p2pMTA* and the corresponding time using SMTP. The conventional system performs better for small messages, but its time increases faster than *p2pMTA* for bigger messages. Higher times in *p2pMTA* are mainly due to the DHT look-ups and the processor-intensive decryption and hashing procedures applied to each message. Also, *p2pMTA* times show the effects of the *GPG* compression: transfer times for text messages are smaller than transfer times for binary messages.

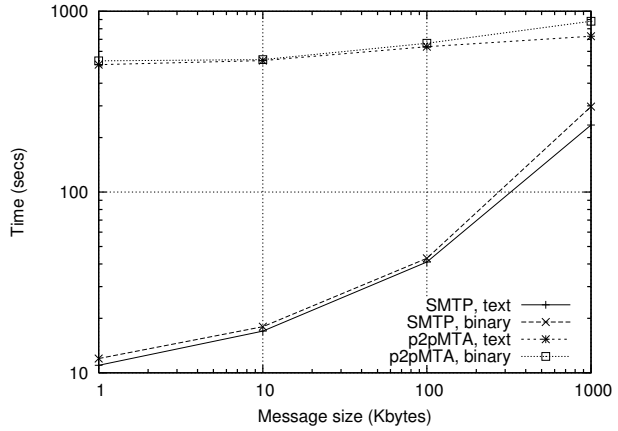

Figure 3. Time to receive 1000 messages

Figure 4 compares the network traffic in both situations. The network traffic plotted is the sum of inbound and outbound traffics on the receiver. Network traffic using *p2pMTA* may be smaller than in the conventional system, because messages are encrypted and compressed. This effect is visible on bigger text messages, because they are more compressible.

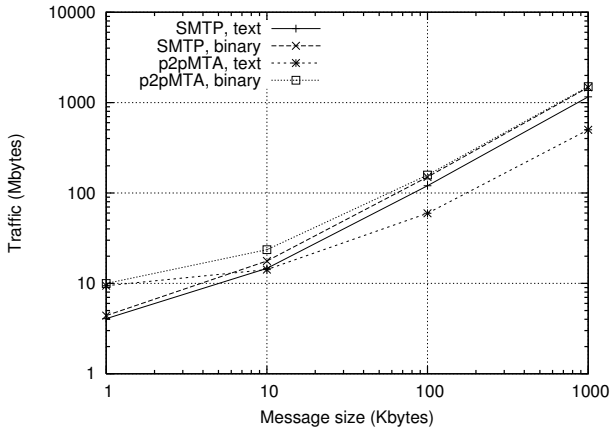

Figure 4. Network traffic to receive 1000 messages

In the previous tests, the receiver accepts all the messages sent to it. If it decides to ignore some of them (because it judges they are spam), its network traffic should reduce accordingly. We evaluate this effect using an experiment in which the sender produces 1000 messages containing a 10K binary file attached. The receiver accepts only a given percentage p of the messages sent to it. Figure 5 presents the network traffic observed in the client according to p, and shows also the corresponding traffic at the receiver side in the SMTP system. The traffic reduction is clear: if the receiver accepts less than 80% of the incoming messages, its network traffic will be inferior to an equivalent SMTP receiver. It should be noticed that the receiver traffic is never zero, because it needs to access the DHT to receive/send notifications.

We are carrying more experiments to evaluate the system scalability and its dependency on the DHT scalability. Also, the cost of the replication mechanism is under evaluation.

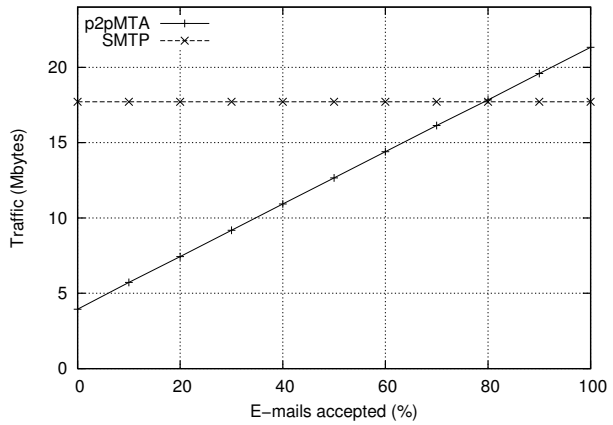

Figure 5. Network traffic at the receiver side, as a function of e-mail accepted

6 Conclusion

In this paper we presented an alternative architecture for e-mail distribution over the Internet. In this architecture, e-mails are stored in the sender side; senders post notifications in a global notification service provided by a distributed hash table. Receivers periodically query it for new notifications; they can then retrieve the corresponding e-mails directly from the senders, or just ignore them. Our architecture got its inspiration from file sharing peer-to-peer systems, in which users locate the resources of their interest and retrieve only the files they choose. Also, messages are replicated among peers to improve their availability.

The main contribution of this architecture is to put the storage overhead on the sender side; in this context, a spammer would fill its own outbox (and the outboxes of its group members) instead of receivers' ones. Also, if a receiver decides to not receive an e-mail, it will not be transferred from the sender, saving its bandwidth. The architecture was implemented in a prototype using COTS components, and some experiments were carried out to estimate its overhead compared with a conventional system. Now we are carrying out experiments to estimate its scalability and robustness.

Some issues remain to be solved or improved. First, peers should be reachable through HTTP (or other transport protocol), because receivers will retrieve their e-mails using it; this may be a problem for peers behind a firewall or NAT router. Also, the integration of this system to conventional e-mail systems should be investigated; a possible solution for this issue is to use *relay* peers to create a bridge between both environments. In our prototype, the peer group is manually defined and static; other methods for managing peer groups should be investigated. Also, messages are fully replicated among peers in a group. A more efficient replication strategy, using techniques as FRS (*Fragmentation, Redundancy, and Scattering* [4, 3]), should be defined. Finally, as each e-mail is replicated, the receiver could retrieve parts of it (fragments) from each group member, to improve retrieval speed; this approach is already used in file sharing services.

References

[1] E. Allman, J. Callas, M. Delany, M. Libbey, J. Fenton, and M. Thomas. DomainKeys identified mail (DKIM) signatures. IETF RFC-4871, 2007.

[2] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4), 2004.

[3] Y. Deswarte, L. Blain, and J.-C. Fabre. Intrusion tolerance in distributed systems. In *IEEE Symposium on Security and Privacy*, 1991.

[4] J. Fraga and D. Powell. A fault and intrusion-tolerant file system. In *IFIP Intl Conference on Computer Security*, 1985.

[5] J. Jung and E. Sit. An Empirical Study of Spam Traffic and the Use of DNS Black Lists. In *ACM Internet Measurement Conference*, 2004.

[6] J. Kangasharju, K. Ross, and D. Turner. Secure and resilient peer-to-peer e-mail design and implementation. In *IEEE P2P Computing*, 2003.

[7] J. Klensin. Simple mail transfer protocol. IETF RFC-2821, 2001.

[8] J. Lyon and M. Wong. Sender ID: Authenticating e-mail. IETF RFC-4406, 2006.

[9] A. Mislove and A. Post. POST: A secure, resilient, cooperative messaging system. In *USENIX HotOS*, 2003.

[10] A. Mislove, A. Post, A. Haeberlen, and P. Druschel. Experiences in building and operating ePOST, a reliable peer-to-peer application. In *ACM EuroSys*, 2006.

[11] L. Peterson, A. Bavier, M. Fiuczynski, and S. Muir. Experiences implementing PlanetLab. In *USENIX OSDI*, 2006.

[12] P. Resnick. Internet message format. IETF RFC-2822, 2001.

[13] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: a public DHT service and its uses. In *ACM SIGCOMM*, 2005.

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

[15] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A bayesian approach to filtering junk E-mail. In *AAAI'98 Workshop on Learning for Text Categorization*, 1998.

[16] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM*, 2001.

[17] M. Wong and W. Schlitt. Sender policy framework (SPF) for authorizing use of domains in e-mail, version 1. IETF RFC-4408, 2006.

[18] Y. Zhao, S. Zhou, and A. Zhou. E-mail services on hybrid p2p networks. In *Grid and Cooperative Computing Conference*, 2004.

[19] P. Zimmermann. *The Official PGP User's Guide*. The MIT Press, 1995.