SureMail: Notification Overlay for Email Reliability

Sharad Agarwal
Microsoft Research

Venkata N. Padmanabhan
Microsoft Research

Dilip A. Joseph*
UC Berkeley

***Abstract*— We consider the problem of silent email loss in the Internet. Some recent studies have reported loss rates of 0.5-1%, which indicates that the problem is significant, especially since email loss can impose a high cost. We present SureMail, a system designed to address the silent email loss problem. SureMail augments the existing SMTP-based email system with a notification overlay to make intended recipients aware of email they might be missing. Our design addresses several challenges including avoiding dependence on support from the email infrastructure, placing minimal demands on users, preventing spammers from subverting the system, and maintaining the privacy of users.**

I. Introduction

The Internet SMTP-based email system does not guarantee the timely or even eventual delivery of messages. Email can sometimes be delayed by hours or days, or even fail to be delivered at all to the recipient(s) [4, 10]. The email sender isn't always notified when such failures occur. Such silent failures, even if rare, impose a high cost on users in terms of missed opportunities, lost productivity, or needless misunderstanding (notwithstanding the purported social "benefit" of plausible deniability offered by email loss). The SureMail system we present seeks to address this problem.

Email could be delayed or lost because of overload, failure (e.g., disk crash), or upgrade of a server along the end-to-end, store-and-forward path from the sender to the recipient. Overload or failure is sometimes triggered by a spurt in the volume of email because of spam or the spread of a virus. Furthermore, the widespread use of spam filters also contributes to email loss by sometimes causing legitimate emails to be discarded as spam. For example, from conversations with the IT staff at a major corporation, we have learned that an estimated 90% of incoming email is dropped even before these hit the user mailboxes or junk mail folders, typically to reduce storage and processing costs for the mail server. Given such an extensive discarding of email, it is hardly surprising that some legitimate email might be caught up in it.

While SMTP does allow servers that cannot deliver emails to auto-generate non-delivery messages to the sender, four issues have reduced the effectiveness of such messages: first, emails dropped by spam filters typically do not cause such messages to be generated; second, spam sent using spoofed source email addresses can generate bogus non-delivery messages to the spoofed source, which can lead to general apathy toward such messages, or worse, classification of such messages as spam; third, some corporations do not allow such messages to be generated to protect the privacy of the corporation (e.g., it prevents an entity from verifying if an email address is invalid); fourth, such responses do not consider emails that are lost between the destination email server and the intended recipient's email client. Thus email loss is often *silent*, such that neither the sender nor the recipient is notified of the delivery failure.

A few recent studies have quantified the extent of silent email loss. Afergan and Beverly [4] report that some mail servers exhibit a silent loss rate of over 5% while Lang and Moors [8, 10] report an overall silent loss rate of 0.69%. Besides these measurement studies, anecdotal evidence suggests that email loss is a non-negligible problem. For instance, consumer ISPs take the trouble to instruct users on what to do when email goes missing (e.g., AOL [1]) and there are companies that offer email monitoring services for businesses concerned about email loss (e.g., Pivotal Veracity [3]). Finally, the authors have themselves experienced and/or are aware of multiple instances of silent email loss recently, including that of a job recommendation letter email sent to Microsoft, a conference program committee invitation email sent from Microsoft, a decision notification email for the IMC 2005 conference sent from a server in Australia, and apparent email loss during a recent server upgrade at UC Berkeley.

Proposals to address the email loss problem have included the message disposition notification mechanism [6](more commonly known as "read receipts") and alternative P2P architectures for email delivery [2, 9]. However, as we elaborate in Section II, these approaches raise privacy concerns and/or are potentially disruptive.

Since the current SMTP-based email system works most of the time, our goal in SureMail is to augment the existing system rather than replace it with a new system of uncertain reliability. SureMail provides a notification mechanism, overlaid on the unmodified SMTP email delivery system, to enable intended recipients to tell when they are missing email. By placing the onus of detecting missing emails on the intended recipient, SureMail preserves the asynchronous operation of email, together with the privacy it provides. SureMail is able to operate with the existing email infrastructure and without requiring a public key infrastructure (PKI) for email users, attributes which we believe aid real-world deployment. The additional reliability of the combination of SMTP-based email with SureMail arises from the orthogonality of the notification overlay (and hence its failure independence) with respect to the email delivery system. Our design includes features that do not make the notification system vulnerable to spam and virus-laden messages as the email system is.

*The author was an intern at MSR Redmond during the summer of 2005.

II. Related Work

Afergan and Beverly [4] report on the state of email health in the Internet, using email bounce-backs as their measurement tool. Based on a study of 1468 mail servers across 571 domains, they found significant instances of silent email loss. For instance, 60 out of the 1468 servers exhibited a silent email loss rate of over 5%, with several others exhibiting a more modest but still non-negligible loss rate of 0.1-5%. They also found instances of emails delayed by more than a day, which might not be much better than email loss from a user's viewpoint. One caveat with this study, however, is that email bounce-backs might not reflect the true health of the email system for normal emails.

Lang and Moors [8, 10] use a more direct methodology to measure email delays and loss. They obtained 40 email accounts across 16 domains and made direct measurements by repeatedly sending emails to these accounts (over 6000 emails to each account over a 3-month period). They report an overall silent email loss rate of 0.69%, with the loss rate being over 4% in some cases. While this study avoids dependence on bounce-backs, there are other biases it may suffer from, including the use of a single sender for all email. [1]

There have been various proposals to address the email unreliability problem, ranging from simple augmentation of the current email system to radical redesign. In the former category is the message disposition notification mechanism [6] (i.e. "read receipts"), which allows a sender to request the recipient (or his/her user agent) to generate an acknowledgment when an email has been read. While many email clients (e.g., Microsoft Outlook) support read receipts, we believe from anecdotal evidence that most users do not enable this feature because it exposes too much private information, viz., how often a user reads email, and whether and when the user read a particular email. Users may find that this exposure conflicts with the inherent "asynchronous" use of email. More importantly, such read receipts — whether sent explicitly through the email system or implicitly through accesses to embedded web content — tell spammers whether an email account is active, thereby making their spamming more "effective".

There have also been proposals to re-architect emails servers or the email delivery architecture itself to improve its reliability [2, 9]. In particular, the POST system [2, 9] proposes a server-less architecture based on a distributed overlay network consisting of the participants' desktop computers. While improving the reliability and availability of email systems is certainly desirable, that alone will not solve the email loss problem because of spam and the resulting filtering of email. To address the spam problem as well as to maintain the privacy of email content, POST assumes a public key infrastructure (PKI) for users, which could be an impediment for deployment. Finally, since POST replaces the current email delivery infrastructure with a new P2P overlay system, it raises uncertainties and risks with regard to its performance if all email were to switch over to the new system.

In contrast, by keeping the notification layer separate from the email system, SureMail ensures that even in the worst case, email performance is no worse than with the current system. We are able to leverage existing work on scalable notification systems (e.g., [5]), so we focus our discussion on the novel aspects of the email problem and our design.

III. Design Requirements

We list here requirements of a solution to email unreliability that we believe will lead to the most rapid adoption.

1) **Cause minimal disruption:** The current email system works for the majority of email. So rather than replace it with a new system of uncertain reliability, we would like to augment the system to improve its reliability. We want to inter-operate seamlessly with the existing email infrastructure (i.e., unmodified servers, mail relays, etc.), with additions restricted to software running on end-hosts. This also eases the deployment of such a system.

2) **Place minimal demands on the user:** The proposed system should minimize demands placed on the user's time. Ideally, user interaction should be limited only to actual instances of email loss; otherwise, the user should not be involved any more than he/she is in the current email system.

3) **Preserve asynchronous operation:** The email system provides a loose coupling between senders and recipients. The sender does not know whether or when an email was downloaded or read by the recipient. Potential recipients do not know whether a sender is "online", i.e., actively sending emails to others. The proposed system should preserve such asynchronous operation, which is in contrast to other forms of communication such as telephony, instant messaging and the use of "read receipts" for email.

4) **Preserve privacy:** The proposed system should not reveal any more about a user's email communication behavior than the current system does. For instance, it should not be possible for someone to determine the number of emails sent/received by another user, the recipients/senders of those emails, how often the user checks email, and so on.

5) **Maintain defenses against spam and viruses:** The proposed system should not make it any easier for spam and email viruses to circumvent the defenses that are in place. All email should continue to be routed through spam filters, so that it does not unnecessarily impose on the user's time. It should not be any easier than at present for spammers to determine whether an email address is valid or to circumvent spam defenses by forging the sender address.

6) **Minimize overhead:** The proposed system should minimize the amount of overhead imposed, in terms of additional traffic and messaging. In particular, we would like to avoid duplicating the work done by the current email system on the data path, i.e., in conveying the message bits from the sender to the recipient.

IV. SureMail Design

We now describe the design of SureMail to satisfy the requirements listed above. We continue to use the current

[1] We are in the process of conducting a measurement study of our own to avoid some of these issues.

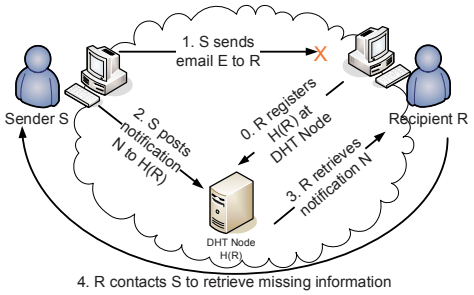

Fig. 1. Overview of SureMail

email system for message delivery. However, we augment it with a separate overlay system for *notification*. The notification system allows the intended recipient of an email to determine if he/she is missing any emails sent to him/her. Depending on the recipient's policy and the identity of the sender, the recipient can choose to request the sender (via email or out-of-band) to resend the lost information/email. Thus SureMail tries to assure senders that either email is delivered or the intended recipients will discover that it is missing.

The basic idea is as follows. When a sender sends an email to a recipient, it also stores a hash of the email contents in a DHT (Distributed Hash Table), indexed by a key derived from the recipient's email address. The recipient looks up the DHT periodically to see if there are any messages that were sent to it. If so, the recipient compares the message hashes obtained from the DHT with hashes computed locally on messages it has received. Any hashes present in the former but not in the latter may indicate missing email.

The DHT system for notifications could be run on dedicated servers and/or on the client computers of the participating users (e.g., office computers that are mostly online). While we are agnostic to the choice of the DHT system, we assume that the DHT nodes support SureMail-specific operations, in addition to the standard $put()/get()$ operations for storing/retrieving $(key, value)$ pairs. It may be possible to leverage a system like OpenDHT [12] for this purpose.

We do *not* assume that the DHT nodes are trusted. In particular, a DHT node could try to spy on a user's email traffic (e.g., learn which senders email a recipient) or generate spurious notifications. SureMail therefore tries to protect against such attacks. Note that spurious notifications are a more serious problem than dropped notifications (which a malicious DHT node can always cause), since the former imposes a cognitive load on users while the latter leaves users no worse off than with the current email system.

While their scalability and availability [7, 12] makes DHTs attractive for SureMail, an alternative would be to have each organization set up dedicated servers to receive notifications for its users and advertise these servers through DNS (akin to MX records for mail servers).

We describe our proposal in more detail below. A key challenge is preventing subversion of the system by bogus notifications generated by spammers or malicious DHT nodes. To address this issue, we first assume the availability of a public-key infrastructure (PKI) for email users. Since a PKI may not be readily available or adopted widely, we then discuss an alternative way of establishing shared secrets between senders and recipients, in the absence of a PKI.

A. Design assuming a PKI

We assume a public-key infrastructure for email senders and recipients. The public-private key pair for a sender S is (S_{pb},S_{pv}) and that for a recipient R is (R_{pb},R_{pv}). We indicate key operations as $Sign(key, value)$ and $Encrypt(key, value)$. We assume a well-known one-way hash function H. All DHT operations are represented using the syntax $func(keyID, \cdots)$, where $keyID$ is the lookup key.

Upon system initialization, each participating email recipient, R, registers with the system by placing a request in the DHT: $RegisterRecipient(H(R), (R, Sign(R_{pv}, R)))$. The DHT node responsible for $H(R)$ [2] verifies the signature using the PKI and then records the fact that R is a registered SureMail recipient. From that point on, any node can post notifications of emails sent to R. However, only R is in a position to present the credentials needed to *read* the notification information. Any notifications posted to an unregistered recipient email address will be discarded, thereby avoiding wasting storage resources on notifications intended for users who are not using SureMail.

When S sends an email E to R, it constructs a notification $N = (H(E), T, Encrypt(R_{pb}, S), Sign(S_{pv}, H(E), T)))$ (where T is the time-to-live (TTL) of the notification) and places a request in the DHT: $PostNotification(H(R), N)$. The DHT node $H(R)$ appends N to the list of notifications for R. It may delete N from the list after the TTL has expired. Since the DHT only provides best-effort service, a sender who is particularly anxious about registering the notification for an (important) email could re-post the notification at a later time. The DHT node, $H(R)$, will either record the notification (if it was missed the first time) or will ignore the repeat posting (since $H(E)$ would identify it as a duplicate).

Periodically, the email client at R queries the DHT: $CheckNotification(H(R))$. The DHT node $H(R)$ performs a standard handshake to authenticate R [3] and then returns to it the list of notifications. The recipient's email client validates the notifications by checking their signatures and then checks the hashes contained in the notifications against hashes of received emails (which can be pre-computed), to determine the ones that may correspond to missing emails. The recipient user is then notified of such notifications, each of which is annotated with the sender's email address. At this point, the user can take corrective action, which could include requesting the sender to resend the missing message identified by $H(E)$. Such a request could be made via email or out-of-band, and with or without human involvement.

[2] In the remainder of the paper, we will use the term "node k" synonymously with "the node responsible for key k".

[3] Although the notifications are encrypted, authentication is desirable to prevent an attacker from gleaning information about the volume of emails to R from the volume of notification information returned. An alternative strategy would be for R to "pad" its notification list with bogus notifications intended to confuse an attacker.

1) Are the design requirements satisfied?: Since the notification system is separate from the email delivery system, the operation of the latter is unaffected by the former. So email performance with SureMail is no worse than with the current email system. Also, the work of carrying the message bits is not duplicated, thus limiting the overhead. Nevertheless, there is the overhead of the DHT operations to post and check notifications, an issue we discuss in Section IV-C.

The asynchronous operation of email is preserved since the posting of notifications by senders is decoupled from the checking of notifications by recipients. Senders do not know whether or when recipients are checking for notifications.

The availability of a PKI facilitates preserving the privacy of notifications and filtering out bogus notifications. Thus the recipient user is notified only when there is missing email and incurs no additional work otherwise. However, the DHT node $H(R)$ is in a position to monitor the volume of notifications posted for recipient R even if the content of the notification and the sender identity are obscured. We discuss possible mitigations in Section V-A.

Finally, since the actual email content is transferred via SMTP, it is subject to checks by the spam and virus defenses already in place.

While this design appears to satisfy all of our design requirements, the dependence on a PKI for email users presents an impediment to adoption. We now discuss a modified design to avoid dependence on a PKI. We also consider ways to reduce the volume of DHT operations and to support mailing lists.

B. Avoiding dependence on a PKI

1) Non-infrastructure based key exchange: Without a public key infrastructure, we can employ a scheme such as PGP (pretty good privacy) or GnuPG. The SureMail enabled client for R generates R_{pb} and R_{pv}, and S generates S_{pb} and S_{pv}. To distribute S_{pb}, S's client needs to attach S_{pb} to the first outgoing email to R, and R does likewise. S and R will have to trust the public keys contained in their first email exchange.

On system initialization, R places a request in the DHT: $RegisterRecipient(H(R), (R, R_{pb}))$. To prevent someone else from masquerading as R and registering on its behalf, DHT node $H(R)$ initiates a simple handshake via email, whereby it confirms that the node purporting to be R can in fact receive email sent to R and decrypt nonces encrypted with R_{pb}. This handshake also helps defend against replay attacks. Once the handshake has completed successfully, the DHT node $H(R)$ stores a record $H(R) \rightarrow R_{pb}$. S posts notifications and R checks for them in the same manner as before.

A problem case for this non-infrastructure based key exchange is when a user S migrates from one SureMail enabled client to another, for example, when S switches to a new computer or a disk crash requires S to setup a new email client. Since there is no PKI that stores S_{pb} and S_{pv}, S needs to somehow obtain these for the new SureMail client.

There are two possible but unsatisfactory solutions. First, the user could be required to backup S_{pb} and S_{pv}, and restore them on new SureMail clients. This is unsatisfactory because it requires user involvement. Second, we could introduce a scheme for migrating keys. Here, if at any time S changes its keys, it attaches the new S_{pb} to its next email to R (or on all emails). R will see the new S_{pb} and migrate to using it instead. A new handshake can also be performed with the DHT node. Unfortunately, this scheme has two drawbacks. First, it requires modification of emails to include the public key. Second, it is vulnerable to an attack where a malicious node T forges email to R as though it were from S, with a bad key, T_{pb}, attached. In this way, T can pollute R's state, invalidating subsequent notifications by S.

2) Reply-based shared secret: To avoid these problems, we need a technique for generating shared secrets between S and R that is both automatable (i.e., does not require direct human involvement) and supports migration. We retain the email-based handshake procedure for registration, as before.

We make the observation that email often involves one user "replying to" a prior email from the other user. The resulting message often has remnants of the original message (in the subject line, quoted text, in-reply-to header [4], etc.). We conjecture that it would often be possible to determine with a low false positive rate whether an email E_2 from R to S is a reply to an earlier email E_1 from S to R. When S receives E_2 and determines that it is a response to the earlier message E_1, S can conclude that E_1 (or $H(E_1)$, for compactness) is a shared secret between itself and the recipient R. Since this state is not known to any other node, S can use $H(E_1)$ to identify itself to R. [5] Similarly, if S replies to E_2 with E_3, then R can subsequently use $H(E_2)$ to identify itself to S.

In this scheme, the notification format that S will use for a new email E_{new} is $(encrypt(H'(E_1), (W, H(E_{new}))), H(E_1), T)$, where H and H' are two different one-way hash functions and W is a string that any recipient can rely on to confirm that the decrypted message is well-formed. Possession of $H(E_1)$ does not permit the malicious DHT node to determine $H'(E_1)$ (the encryption key) and hence post a fake notification. While a malicious DHT node could replay old notifications, these are easy to filter out. When R retrieves a notification that has not expired, it looks up $H(E_1)$, identifies the sender S, and uses $H'(E_1)$ to decrypt the notification. It then checks $H(E_{new})$ against the hashes of previously received emails from S to determine if it corresponds to a missing email.

For a first-time sender who has not established a shared secret with R, there are two choices — either post an unencrypted notification to the DHT, thereby revealing his/her identity to the DHT, or not post a notification at all in the hope that email loss does not occur. When R receives this notification, it will place this notification in the "Junk notifications" category, according it lower priority than notifications from "trusted" senders (i.e., senders from whom R has received emails and to whom it has sent emails in the past). In fact, it is fundamentally hard to distinguish between a first-time

[4] However, message-ID of the original email may be inserted by the email server and hence not be readily available to the sending client.

[5] To be safe, S should only pick email, E_1, whose only recipient was R.

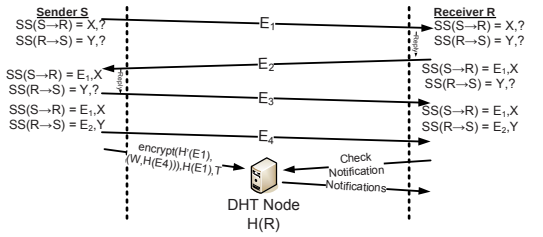

Fig. 2. Shared secret (SS) evolution between S & R in reply-based scheme

legitimate sender and a spammer. Note, however, that this would be a problem only in the event that the first email from a sender to R is lost. [6]

In terms of state, R needs to remember two hashes for each email address it corresponds with, one for sending and one for receiving. For better security, and to handle the case of migrating a user to a new installation of the email program, the values of E_1 and E_2 can always be refreshed to the last pair of emails exchanged between R and S, as follows (and as shown in Figure 2). S remembers the hashes of all messages sent by it since the most recent one replied to by R. Likewise, R remembers the hashes of all emails from S (which it had replied to) since the most recent email that S has used as shared secret in a notification.[7]

Thus a user who migrates to a new client can automatically establish a new shared secret with a peer based on fresh email exchanges. This shared secret is then used to authenticate fresh notifications. The only window of vulnerability is the time between when the user migrates and when a new shared secret is established. The user may not be in a position to authenticate notifications posted during this window or post any authentic notifications itself.

Since establishing a shared secret between a sender and a recipient requires an email from the former to be replied to by the latter, an attacker is prevented from polluting this state, unlike the case of a non-infrastructure-based key exchange. A bogus "reply" generated by the attacker will be rejected since the sender would have no record of having sent the original email that the attacker's email purports to be the reply to.

The reply-based shared secret scheme has the advantage that it leaves the emails themselves untouched. However, unlike non-infrastructure-based key exchange, it is susceptible to eavesdropping. An attacker who is able to listen in on email communication to/from a user can learn the reply-based shared secrets of the user, and so would be in a position to post bogus notifications purporting to be from senders that the user trusts. However, such an attacker would also compromise the user's privacy, which is arguably a greater threat.

3) Are the design requirements satisfied?: Compared to the PKI case, the assurances provided with the reply-based shared secret scheme are somewhat weaker but still quite good. The design without a PKI hinges on (a) the inability of an attacker to eavesdrop on or intercept the registration or authentication messages exchanged with a recipient, to prevent the process from being hijacked, (b) the secrecy of emails exchanged between two users, and (c) the ability to detect email replies to set up the shared secret. We believe that all of these assumptions are reasonable in practice. If (a) or (b) were not to hold, then the privacy of email would be compromised anyway, regardless of whether there is a notification system. Regarding (c), we have prototyped a reply detection algorithm based on measures of text similarity and our initial results are promising. Note that we only want to avoid false positives but can tolerate false negatives so long as at least some replies are detected to allow the shared secret to be updated.

[6]While these events could be correlated, we assume this is rare.

[7]The temporal checks in this adaptive scheme could be relaxed a little to account for clock skew across multiple clients that the sender and/or recipient might be using.

C. Reducing DHT Overhead

Since the SMTP-based email infrastructure works fine most of the time, posting a notification in SureMail for every email sent could add up to significant and unnecessary overhead. To reduce overhead, a sender could hold off on posting a notification for a while, in the hope that an "ACK" in the form of a reply or a "NACK" in the form of a bounce-back (both detected automatically, without human involvement) would obviate the need for the notification. Notifications for multiple messages to a recipient could also be coalesced to reduce overhead. Finally, a sender or his/her email client could chose to post notifications only for a small subset of messages that are deemed important enough to be protected using SureMail. For example, these might be emails sent to particular recipients, ones with the "Importance: high" flag set, etc. As a data point, from a a sample of 57,972 emails drawn from half-dozen colleagues at Microsoft, we found that only 2.3% had the "Importance: high" flag set.

Another optimization is for the sender to include hashes of previously sent emails in new emails. This procedure can possibly help detect email loss quickly, obviating the need for a notification to be posted to the DHT but at the cost of modifying emails. Such loss detection would work only when there is sporadic loss in an otherwise steady stream of emails.

D. Supporting Mailing Lists

Notifications are posted as usual for emails sent to mailing lists. The only difference is that the optimizations listed above involving holding back on notifications would not apply. The notification for an email sent to mailing list M would be sent to $H(M)$. It would be the responsibility of individual recipients to look up the DHT periodically for notifications of emails sent to mailing lists that they are members of.

V. Discussion

A. Attacks on SureMail

Despite the steps taken by SureMail to protect the privacy of both senders and recipients, an attacker could try indirect means to glean information. For instance, a (trusted) sender could post spurious notifications with carefully controlled TTLs and then infer a recipient's email checking (or at least notification checking) habits based on which emails, if any, the recipient asks for a retransmission of. However, a recipient could watch out for and choose to ignore suspicious patterns

of notifications (e.g., frequent notifications from a sender indicating apparent email loss).

A DHT node could use knowledge of the IP addresses from which notifications are being posted for each recipient to reverse-engineer some information about the senders. This threat could be alleviated by passing the notification through a forwarding chain (as in standard DHT routing or in anonymous communication systems such as Crowds [11]) instead of directly sending it to the $H(R)$ node.

Since the DHT node $H(R)$ authenticates the request for notifications from R (whether using a PKI or an email-based handshake), it is in a position to monitor the volume of notifications intended for R, even if it cannot tell the identity of the senders. To alleviate this problem, R could register with a different DHT node as a function of time (say node $H(R, d)$, where d is the current date/hour), thereby denying any single DHT node the opportunity to monitor its notifications on a continuous basis. Alternatively, registration/authentication could be decoupled from the posting of notifications. While notifications are posted to node $H(R)$ as before, R registers itself with node $H(H(R))$, which allows another node (say $H(R)$) to authenticate R without learning its identity. We assume that only a small fraction of the nodes in the DHT is malicious and/or in collusion.

B. Should emails themselves be delivered via SureMail?

Delivering emails themselves through the SureMail overlay is problematic for several reasons. First, emails are typically much larger than notifications, so transporting them through the overlay incurs a much larger overhead. Second, it is still desirable for emails sent via the overlay to be routed through spam filters, virus scanners, etc., which leaves open the possibility of email loss. In contrast, notifications are fixed-sized, fixed-format entities that do not pose the same threat as malicious email content and hence need not be filtered the same way. Finally, as noted in Section II, using the overlay for delivering the emails themselves is potentially disruptive and runs the risk of under-performing the current email system.

C. Should SureMail be integrated with email infrastructure?

We consider whether SureMail's notification mechanism should be integrated directly into the existing email infrastructure, for example, by having the receiving email server or spam filter generate notifications locally for emails that are dropped.

We believe that there are several reasons why such integration could be problematic and so having SureMail operate separately would be advantageous. First, since the email server by itself cannot distinguish dropped emails originated by senders that the recipient trusts from other email (e.g., spam), the server is forced to generate notifications (including computing a hash of the message content) for *all* dropped emails. This can place a significant burden on the email server, with potentially negative impact on normal email delivery. Second, even if load were not an issue, a recipient's email server cannot by itself generate notifications that allow the recipient to distinguish between dropped emails from trusted senders from other dropped emails, resulting in burden on the user to sort through them. Avoiding this problem would require support at the sender end, as in SureMail. Third, often spam is dropped without even looking at the email content, e.g., by blacklisting certain IP addresses. According to the IT staff of a major corporation, over 98% of dropped emails are estimated to be due to such steps. It is impossible for the email server to generate meaningful notifications in such cases. Finally, to the extent that email loss happens not because of a conscious decision to drop email but due to failure, it would be hard for the (failed) infrastructure to generate meaningful notifications.

Thus we believe that keeping the SureMail notification layer separate is advantageous since it avoids burdening the email delivery infrastructure and is less prone to correlated failures of both the email delivery and notification systems. We also believe that placing the burden of ensuring reliable email delivery on the senders and recipients themselves rather than the email delivery infrastructure is in conformity with the end-to-end principle [13] that underlies the Internet's design.

VI. Conclusion

We have presented SureMail, a system to augment the current email delivery infrastructure with a notification overlay to notify recipients of email loss. We have shown how SureMail can operate without requiring support from the email infrastructure and without requiring a PKI for email users. We are in the process of conducting a measurement study of email loss as well as prototyping SureMail. Our initial implementations of automatic reply-detection and bounce-back detection are promising.

Acknowledgements

We thank John Dunagan, Dave Maltz, and Dan Simon for their feedback.

References

[1] AOL Member Missing Email Self Help Page. *http://postmaster.info.aol.com/selfhelp/mbrmissing.html*.

[2] ePOST Serverless Email System. *http://www.epostmail.org/*.

[3] Pivotal Veracity: Tools & Intelligence for Optimizing Delivery. *http://www.pivotalveracity.com/*.

[4] M. Afergan and R. Beverly. The State of the Email Address. *ACM CCR*, Jan 2005.

[5] L. Cabrera, M. Jones, and M. Theimer. Herald: Achieving a Global Event Notification Service. *HotOS*, May 2001.

[6] R. Fajman. An Extensible Message Format for Message Disposition Notifications. *RFC 2298, IETF*, Mar 1998.

[7] M. Freedman, E. Freudenthal, and D. Mazieres. Democratizing Content Publication with Coral. *NSDI*, Mar 2004.

[8] A. Lang. Email Dependability. Bachelor of Engineering Thesis, The University of New South Wales, Australia, Nov 2004. *http://uluru.ee.unsw.edu.au/~tim/dependable_email/thesis.pdf*.

[9] A. Mislove, A. Post, C. Reis, P. Willmann, P. Druschel, D. Wallach, X. Bonnaire, P. Sens, and J. Busca. POST: A Secure, Resilient, Cooperative Messaging System. *HotOS*, May 2003.

[10] T. Moors. Email Dependability. *Email Management World*, Aug 2004. *http://uluru.ee.unsw.edu.au/~tim/dependable/email/emw_moors.pdf*.

[11] M. K. Reiter and A. D. Rubin. Crowds: anonymity for Web transactions. *ACM Trans. on Information and System Security*, 1(1):66–92, 1998.

[12] 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. *SIGCOMM*, Aug 2005.

[13] J. Saltzer, D. Reed, and D. Clark. End-To-End Arguments in System Design. *ACM Transactions on Computer Systems*, 2(4), Nov 1984.