Spam-I-am: A Proposal for Spam Control using Distributed Quota Management

Hari Balakrishnan and David R. Karger
MIT Computer Science and Artificial Intelligence Laboratory
The Stata Center, 32 Vassar St., Cambridge, MA 02139
Email: {hari,karger}@csail.mit.edu

ABSTRACT

Email spam has reached alarming proportions because it costs virtually nothing to send email; even a small number of people responding to a spam message is adequate incentive for a spammer to send as many messages as possible. Since spammers need to send messages at high rates to as many recipients as they can, *quotas* on email senders could throttle spam. We argue for separating the *allocation* of quotas, a relatively rare activity, from the *enforcement* of quotas, a frequent activity that must scale to the billions of messages sent daily.

This paper tackles the quota enforcement problem, where the goal is to ensure that no sender can grossly violate its quota. The challenge is to design an enforcement scheme that is scalable, is robust against malicious attackers or participants, and preserves the privacy of communication, in a large, distributed, and untrusted environment. We discuss the design of such a system, *Spam-I-am*, based on a managed distributed hash table (DHT) interface, showing that it can be used in conjunction with electronic stamps (for quota allocation) to ensure that any non-negligible reuse of stamps will be detected.

1. INTRODUCTION

I do not like that spam in RAM
I do not like it, Spam-I-am.
With apologies to Dr. Seuss

The email spam problem has reached alarming proportions—some studies estimate that spam accounts for nearly 60% of the over 50 billion email messages sent daily [5, 15]. To combat spam, tools such as SpamAssassin [19], which filter email based on message content, have become popular. Unfortunately, such tools are routinely overwhelmed by the creativity of spammers in getting past the filters; when users set more aggressive filter thresholds in response, legitimate email is trapped as spam.

Attention has therefore been given to schemes that force the sender to consume some resource, such as computation, money, or human effort, in order to send email [1, 6, 9, 2]. If resources must be consumed to send email, and individuals have limited resources, then the number of messages they can send is limited. Because few legitimate users send email at the sheer scale of spammers, such a *quota* on every user (legitimate or not) could effectively choke off spam without limiting legitimate email.

A quota-based spam control system must tackle two distinct issues, which we argue must be treated separately. The first, *quota allocation*, is determining how quotas are allocated to individuals. The second, *quota enforcement*, is providing a way to detect and respond to quota overruns.

We propose a spam control system called *Spam-I-am* based on explicit email quotas. Spam-I-am separates quota enforcement from allocation, letting us optimize both pieces: investing great care in allocation so that spammers cannot cheat to receive large quotas, while offering an enforcement system running on a cheap, large-scale, distributed, untrusted infrastructure, and scaling to meet the significant demands of having to verify the validity of every single received email message. This infrastructure is based on a distributed hash table (DHT) [13] and does not require a large resource allocation by any one party.

Spam-I-am requires that an email user (or user's organization) contact a globally trusted *quota allocator* once per year (say), and obtain a quota in exchange for payment.[1] This quota is a digitally signed and dated certificate of the user's right to send a certain number of messages. It is signed by the allocator, whose public key is widely known. The user's outgoing mail server (or email sending program) manufactures a quota-limited number of unforgeable *stamps* using standard cryptographic operations, and attaches one to each outgoing message. Spam-I-am enforces the quota by helping any recipient determine if a stamp is authentic and fresh. If a received stamp is both authentic and fresh, the recipient knows that the message was within the sender's rightful quota. If not, the recipient may discard the message (or take some other action, as described at the end of this section).

Spam-I-am's enforcement infrastructure for verifying whether a stamp is fresh is a collection of untrusted machines acting as a *distributed quota manager* (DQM). A DQM stores all stamps used in some past time window. A

[1]The payment is not crucial; any suitable means of verifying the user's right to a quota would suffice. See Section 5.

recipient queries DQM to determine if a stamp is fresh, while at the same time informing DQM, with suitable proof, that it has received a message with a given stamp.

Spam-I-am meets the following key design goals:

Scalability. The system should scale to hundreds of millions of users and tens of billions of daily messages, requiring only modest resources at each node.

Robustness. The system should resist senders who try to violate their quotas, subversion by malicious participants, and denial-of-service (DoS) attacks. Faults should never interfere with the delivery of legitimately stamped email, and no fault should let more than a small amount of spam through.

Privacy. DQM nodes should not be able to infer who is communicating with whom. In addition, a recipient should not be able to prove that the sender sent it any email, and the sender should not be able to prove that the receiver received it.

User-transparency. The system should not require users to expend additional effort to send (*e.g.*, as with CAPTCHAs [21]) or receive email.

The DQM infrastructure avoids many problems faced by systems that use a central, trusted quota enforcer [1, 14]. A central enforcer needs to scale to meet the demands of the global email system, imposing significant resource requirements on its operator and constituting a single point of failure. In particular, individuals who wish to violate their quotas may mount attacks on the central enforcer. It is also unclear whether offering a centralized enforcement service is a profitable business; and if it were, it is unclear how competing enforcers would interact and how ethical each would be. Instead, we envision that DQM will be composed of servers contributed by participating organizations (*e.g.*, ISPs, companies, universities, etc.), managed like the Internet's email and Domain Name System (DNS) infrastructures.

Unlike the enforcement system, Spam-I-am's quota allocator requires little state or computational power because user interactions with it are rare. Since the allocator is not used in enforcement, and users can proactively renew quotas, outages in the allocator caused by faults or attacks will not seriously affect email delivery. The main requirement for the allocator is that it be trusted, a property many nonprofit organizations possess.

We expect that Spam-I-am would typically be used in combination with passive spam filters. Most spam filters have a tunable "level of paranoia" to balance false positives against false negatives. Spam-I-am provides a second chance for legitimate email accidentally blocked by filters: any such email can be passed with a legitimate stamp. This extra protection from falsely blocked email will let users make their filters much more aggressive, thus blocking a larger fraction of true spam, even before Spam-I-am is fully adopted.

2. RELATED WORK

Dwork and Naor proposed that each message come with a signature that takes significant CPU time [9] or memory [8] to compute. Microsoft's Penny Black proposal [1, 17] uses a centralized and trusted ticket server to allocate tickets to clients based on such "proof-of-work". Email recipients contact the ticket server to validate tickets. Camram [6] uses a similar approach to have email senders attach "hashcash" [2] stamps to their transmissions. Laurie and Clayton [15] have argued convincingly that proof-of-work schemes are unlikely to control spam because of the ease with which spammers are able to subvert numerous insecure computers on the Internet.

Various systems, including Bonded Sender [3], Vanquish [18], and SHRED [14] have proposed that email senders should forfeit money, or be sued (Habeas [12]), if a recipient deems a message to be spam. Such a "bond" [3, 18] or "contingent liability" [14] requires users to mark up the email they receive, or carefully tailor their spam filters.

The Sender Policy Framework [20] verifies if the sender email address is consistent with information published in DNS about valid email servers in the sending domain. It provides some protection against forged sender addresses, but none against spammers using unforged addresses.

Spam-I-am's stamps and DQM infrastructure bear some similarity to digital payment schemes [10]; quota allocation corresponds to digital cash withdrawal, stamps correspond to spending a small amount of digital cash to send every message, and quota enforcement corresponds to forgery prevention and double-spending detection. However, the looser requirements of spam blocking —recipients do turn in their stamps for real cash, and a small amount of double spending does not hurt—permit a simpler design. Conversely, our approach may be useful for digital cash schemes, allowing forgery and double-spending to be detected in distributed fashion.

3. SPAM-I-AM DESIGN AND PROTOCOL

Spam-I-am's protocol is underpinned by a trusted quota allocator QA with a public/private key pair (QA_{pub}, QA_{priv}).[2] A participant S constructs a public/private key pair (S_{pub}, S_{priv}) and presents S_{pub} to QA together with either payment or some accepted form of identity. After determining that S should be allocated a quota, QA gives S a signed[3] *certificate* $C_S = \{S_{pub}, \text{expiration time}\}_{QA_{priv}}$ indicating that a quota has been allocated, where "expiration time" is the time at which the certificate C_S expires (typically one year or so). Anyone knowing QA_{pub} can verify that S has been allocated a quota. It is possible to have several independent (but globally trusted) quota allocators.

[2]Only QA_{pub} needs to be widely known; Spam-I-am does not require a public key infrastructure (PKI).

[3]We denote signing operations as subscripts on messages.

For quota enforcement, Spam-I-am's DQM requires a (possibly faulty) "hash-table" interface, offering the operations $\text{PUT}(k, v)$ to associate value v with key k, and $\text{GET}(k)$ to determine the value previously associated with key k. We also use a hash function, such as MD5 or SHA1, which we assume every participant can compute but no participant can invert.

3.1 Basic Protocol

We introduce our approach using a simple example in which we wish to limit each user to sending m distinct email messages. Sender S includes an unforgeable *stamp* with each message sent to recipient R. Upon receipt of the message, R checks with DQM whether the included stamp has been used before by querying for an unforgeable *postmark* that is used to "cancel" that stamp. R only reads the message if the postmark for its stamp is not published in DQM. R also constructs and publishes the stamp-cancelling postmark in order to prevent the stamp from being reused later. It follows that each stamp can be used to send exactly one message (that is read).

The sender S uses its private key to construct m stamps of the form $\{C_S, i\}_{S_{priv}}$, where i is an integer between 1 and m. We call i the *counter*; the intent is that each counter value be used in a stamp exactly once. Note that only S can construct its stamps since only it can sign with S_{priv}. However, any recipient can check that a stamp is valid by verifying the signature with S_{pub}, contained in C_S, confirming that $1 \leq i \leq M$, and then consulting DQM to test if the stamp is fresh or not.

R cancels stamp P by executing $\text{PUT}(\text{HASH}(P), P)$; *i.e.*, R stores the stamp as the *value* associated with the stamp's postmark (the key), $\text{HASH}(P)$. To prevent prevent malicious participants from cluttering DQM, DQM accepts this PUT only if the value is a valid stamp and the hash of the value equals the key. Conversely, to decide whether to read the message, R queries the postmark by invoking $\text{GET}(\text{HASH}(P))$ to determine whether P was previously cancelled. R blocks the mail as spam only if the correct value (P) is returned as the result of its GET.

3.2 Security

This basic protocol has three key security properties. First, no mail with a legitimate fresh stamp can ever be blocked as a result of the protocol, regardless of malicious third parties or DQM nodes. Second, when DQM operates correctly, no malicious third party (spammer) can prevent used stamps from being cancelled. Third, each fault in a DQM node allows only a limited number of reuses of cancelled stamps.

We first argue that no legitimate mail can be blocked by our protocol. Since it is signed, a stamp cannot be generated by anyone other than the sender. Furthermore, since the hash function is hard to invert, no one but a recipient of stamp P—not even a malicious DQM node, upon receipt of the query $\text{GET}(\text{HASH}(P))$—can generate a postmark for it. In other words, the only way a cancelling postmark will be found is if some entity received the corresponding stamp and cancelled it.[4] Now recall that when a recipient fails to locate a valid cancellation, they accept the mail as legitimate. Since we have just argued that such a cancellation cannot exist for a fresh stamp, we can conclude that no mail carrying a fresh stamp will be blocked.

We now consider the converse attack—ways that spammers might attempt to reuse their stamps by preventing cancellations. Since recipients ignore invalid cancellations (in case DQM is corrupted), an attacker might hope to insert an invalid cancellation of their stamp. However, since DQM verifies all PUT operations, no incorrect value can be associated with a key. Thus, no third party (spammer) can prevent a stamp from being cancelled by inserting an invalid postmark for it.

Beyond cheating in the protocol, spammers may attempt to interfere with its execution using DoS or subversive attacks on DQM. We discuss such attacks in Section 4.

If a sending account is compromised and commandeered to send spam, it can send to at most m emails before its stamps are exhausted. At this point, the compromised account's users will suddenly discover that their legitimate email transmissions are being bounced due to cancelled stamps. These bounces will provide a strong hint that the machine is compromised, and an equally strong incentive to fix the problem. Our approach moves the cost of being hacked to the person being hacked, rather than to the multitude of email recipients. Once the owner of the affected machine has disinfected the machine and received new certificates for the senders using the machine, the affected senders can resume sending email.

3.3 Privacy

As presented thus far, the protocol violates some communication privacy because the sender's identity, as part of the stamp, is published in DQM. Revealing the sender's identity in the stamp allows DQM (or any eavesdropper) to determine that the sender is sending some email, and possibly connect that email to the recipient performing the query. To eliminate this violation of privacy, we note that DQM is needed only to bind postmarks to stamps; *DQM does not need to understand the contents of a stamp*. Thus, instead of using the information-carrying stamp P as the value in DQM, we can use $\text{HASH}(P)$. With this method,

[4] If email is not sealed, then nodes in the relaying of the path between S and R can act as "recipients," cancel stamps, and cause email to be flagged as spam. But this problem already prevails under today's email design where nodes en route can usually modify or drop messages. To preclude this attack, S could seal the message with R's public key, obtaining that in some fashion. Our system may increase the motivation for spammers to "hijack" email, dropping the body, and taking the stamp for their own use, so such sealing may become more important.

R executes $\text{PUT}(\text{HASH}(\text{HASH}(P)), \text{HASH}(P))$ to cancel stamp P. Since the recipient receives P, it can construct both key and value, but DQM cannot.

Figure 1 summarizes this improved protocol.

1. $\text{STAMP} = \{C_S, i\}_{S_{priv}}$
2. $S \rightarrow R : \{\text{STAMP}, msg\}$.
3. Upon receiving a message, R verifies using S's public key from C_S that STAMP is valid. If it is not, then R discards the message. Otherwise, R computes POSTMARK = $\text{HASH}(\text{HASH}(\text{STAMP}))$.
4. $R \rightarrow$ DQM : $\text{GET}(\text{POSTMARK})$. R considers STAMP used only if $\text{HASH}(\text{STAMP})$ is returned.
5. $R \rightarrow$ DQM : $\text{PUT}(\text{POSTMARK}, \text{HASH}(\text{STAMP}))$.

Figure 1: Spam-I-am's privacy-preserving protocol.

This modified protocol prevents DQM (or anyone else without access to the stamp) from associating the sender and recipient. The recipient, by exhibiting the stamp, can prove that the sender sent *some* message to *someone*. However, since the stamp is not determined by the message or recipient, the recipient cannot prove what message what sent, or to whom.[5] If the sender randomizes the order of his counter, the recipient cannot even draw conclusions about the number of messages sent by that sender.

Conversely, the sender, by checking for a postmark, might become confident that the the receiver received the message; however, since intermediate relays (or the sender itself) can cancel the stamp, the recipient can plausibly deny having received the message.

3.4 Expiring stamps

Spam-I-am's stamp protocol requires DQM to remember cancelled stamps for as long as the sender's certificate is valid. If a postmark is dropped from DQM, anyone can reuse the corresponding stamp. However, remembering cancelled stamps for that long could be a significant undertaking—it requires a significant quantity and robustness of storage.

Our solution to this problem is to use stamps that *expire* (become unusable) within a small number of days, say a week,[6] at which point their postmarks can be dropped from DQM. We augment each stamp with a *weekstamp*, a continuously increasing integer indicating the particular week during which the stamp is valid. The stamp thus becomes $\{C_S, i, w\}_{S_{priv}}$, where w is a weekstamp. Recipients can easily check that stamps are being used in the right week, but the weekstamp serves as a nonce that prevents anyone else from generating valid new stamps from the stamps used in past weeks. The rest of the protocol is the same as in Figure 1.

Since we cannot hope to precisely synchronize all participants, and since email may be delayed in delivery (but probably not by more than a week), the recipient should accept stamps from the previous week as well. Even so, DQM needs to hold on to stamps from the current and previous week only, dramatically reducing its storage and reliability requirements.

Note that the easier alternative of making *certificates* expire every week forces frequent (and therefore undesirable) interactions with the centralized quota allocator.

4. A DHT-BASED DQM

The DQM protocol involves huge numbers of independent hash-table queries, and is naturally implementable over a DHT [13]. DHTs provide the PUT/GET interface required by DQM, assigning each key (postmark) to a particular machine in the network, and providing a scalable and dynamic routing protocol that lets any machine locate and query the node responsible for a given key.

A rough calculation suggests that each postmark-stamp pair requires 32 bytes, and that each insertion will generate roughly 100 bytes of PUT and GET traffic. Assuming that roughly 10^{11} emails are sent daily [15], the average bandwidth for verifying this volume of email is about 1 Gbit/s, and the packet rate is about 2 million packets/s. The peak rates may perhaps be 10 or 20 times higher. We would need to store roughly 10^{12} stamps (two weeks' worth), for a total of about 30 terabytes of storage. While these numbers might be daunting for a single machine, this load could easily be handled by a DHT of 2000-10000 machines, each storing 6-30 GBytes of postmarks and serving at an average rate of 100 Kbits/s to 500 Kbits/s. This DHT could be an open DHT platform such as [16], or could be bundled with SMTP servers.

4.1 Security

We have previously considered attacks on the protocol; we must also consider attacks on the DHT that implements it. We do *not* attempt to solve the problem of attacks on the DHT *routing* layer—that is a subject of much current work within the DHT community [7]. We focus our attention on the *DQM-specific* attacks that adversaries may make.

For each key, the DHT implicitly specifies a *routing tree* through the DHT nodes, rooted at the node responsible for the key. For load balance, this tree is of low degree—typically, degree $d = O(\log n)$ for an n-node system.

Suppose that the adversary is able to crash or compromise some of the machines. Although DHTs cope well with crashes, subversion is another matter. As discussed before, no action by DQM can block receipt of legitimately stamped mail; however, an adversary might hope to use their compromised DQM machines to let their own

[5]Schemes that hash the message to produce a stamp would violate this property.

[6]This interval is chosen because of the particulars of email; it results in reasonable storage requirements, copes with email delays, and permits burst email transmissions.

spam get through; *e.g.*, by failing to respond correctly to queries about cancelled stamps.

Caching can cope with such adversarial attacks, at the expense of increased storage. The DQM nodes along the DHT path from the query origin to the root responsible for the key can cancel the stamp at every node on the query path. A node making a postmark query can stop anywhere along the path, as soon as it gets a positive response.

With this approach, an adversary cannot get their mail through by sending so many copies of a stamp that the machine responsible for holding its postmark gets swamped by the recipients' queries. Instead, each node will receive only one query for a given postmark routed through each of its children, for a total of at most d queries, before its children all know the answer and insulate it from future queries.

A similar argument applies to adversarial DQM nodes. Any node may choose to maliciously answer "no" to a postmark query instead of "yes." However, a response of "no" will just lead to the receiver continuing along the routing path, querying other nodes. Thus, the effect of a false "no" is to exclude the malicious node from the DHT, slightly increasing the query-answering load on some other nodes. The only exception to this argument is at the root for a key: if the root answers "no," then the recipient believes the mail to be legitimate. But because the recipient cancels the stamp all along the query path, each time the malicious root node answers "no", another of that root's children caches that cancellation. Using this argument, we can show that if k machines are compromised but $n > kd$, the number of additional spam messages accepted by recipients is small. We can further improve robustness by cancelling stamps and querying for postmarks at multiple roots, at the expense of increased communication.

Another possible attack on the DHT might come in the form of an exhaustion attack, in which the adversary tries to fill the DHT with so many bogus, cancelled postmarks that the DHT has no room to store the valid ones. If we store the stamps explicitly rather than as a hash, then the adversary cannot mount this attack because they cannot generate valid stamps. The price we pay for email privacy is that DQM can no longer tell from looking at a stamp whether it is real. Our solution to this problem is to use the DQM idea recursively, treating DQM storage as the resource whose quota should be enforced per-recipient. We reserve the details for a longer version of this paper.

5. ALLOCATING EMAIL QUOTAS

We now consider various mechanisms for allocating quotas. Simply throttling the number of daily messages sent per user is not effective because of the ease with which spammers can create new accounts. But several approaches do show promise. One possibility is to use "proof-of-work" for allocation, and use Spam-I-am's DQM to enforce quotas. Yet another approach would be some form of trust network rooted at the central server. A third would involve some government agency—a user could pick up their key in person, while renewing their driver's license or passport. Another is to require payment, whose value depends on the statistics and economics of spam. This section explores this last possibility—our goal is to determine a suitable price for email that would cut spam by a factor of ten.

The Radicati Group estimated that in November 2003, 57 billion emails were sent daily [15]; at the same time, Brightmail estimated that 56% of all email was spam [5]. That means that there were about 32 billion spam messages daily (it is likely that the number is larger today). The same survey estimated that spammers took in roughly US $10 million/day in email revenue, or roughly 0.03 cents per message. This number is consistent with published estimates of rates charged by spammers [11, 4].

We now make an unjustified assumption. Given the ease with which spammers can send bulk email today, they must already be receiving as much income as marketers are willing to pay them, regardless of the quantity of email sent—if there were more demand, it could easily be met. We therefore take $10 million/day as an upper bound on spammers' total "budget" available to spend sending email (this estimate is generous, since it assumes that spammers have no operating costs). If we charge for quotas at a rate of $3 for 1000 messages, then spammers on that budget can afford to purchase at most 3 billion messages per day—a factor of 10 reduction in current rates. We remark that the two authors sent less than 13,000 messages each in the past year, which would have cost them less than $40.

Should this per-email charge seem high, proof of payment to some other institution could be used. A promising possibility is donation to a known-legitimate charitable organization of the sender's choice. With this approach, email no longer "costs extra."

6. OTHER ISSUES

6.1 Stamp reuse

Most legitimate users send email in their "social network" of friends, colleagues, and acquaintances, and every once in a while to someone new. In contrast, a spammer benefits from new recipients reading his messages. These observations suggest that the ability to *reuse* stamps in conversations that have already been established between a sender and receiver could reduce the number of stamps needed by most users, reducing their costs. The quota then becomes the number of *distinct recipients* that a sender can contact. (The two authors each sent messages to less than 2,000 distinct recipients in the past year.)

To allow stamp reuse, each recipient simply needs to remember which senders (certificates) initially contact it with a valid stamp, and continue to accept email from those senders with those (cancelled) stamps. We continue to require the stamp, but we use it only for identification to prevent sender spoofing. We also need to prevent replay attacks, for which standard techniques suffice.

One way in which a spammer might subvert stamp reuse is to first send legitimate-looking email to a recipient, get his stamp "whitelisted," and then send spam using that stamp. Similarly, when a sender's machine is compromised, recipients that have approved the sender's stamp can become targets for large quantities of spam. To solve this problem, a recipient's spam filter should cancel a stamp if there is even a small suspicion of spam. In addition, the sender should check if a previous stamp can be reused before sending a message with the stamp.

6.2 Mailing lists

Mailing lists pose problems because using a different stamp for each recipient may be too expensive, especially for large lists. When mailing lists are managed or moderated, DQM provides sender verification that can be used to pass through email from the mailing list. For large, unmoderated mailing lists with no control over who can send, our approach does not alleviate the problem. A single message within the sender's quota can reach a huge number of recipients. Such lists are a boon to spammers.

6.3 Mail forwarding

Some users may wish to forward their mail to one or more other addresses. Since the first check for a stamp's freshness would also cancel the stamp, the subsequent hops may wrongly consider a legitimate message to be spam. An easy solution is for each hop to forward the message with authentication, so the subsequent hops know that the stamp has already been validated. Another possibility is to use a "fresh" stamp for each hop. With stamp reuse, this approach uses few additional stamps.

7. CONCLUSION

Quotas can prevent the large-scale abuse of resources in a distributed system. With quotas, the system designer wishing to prevent resource abuse has two issues to resolve: how to allocate quotas, and how to enforce them. In the context of email, the primary resource being abused is recipient time, for which the quota should control the number of emails sent per sender. This paper outlined a scheme for scalable, privacy-preserving distributed quota enforcement, and suggested ways in which email quotas could be allocated. Our approach could reduce spam by at least one order of magnitude, for a modest annual payment per user. The general approach is also likely to be useful in controlling spam in other systems, such as instant messaging and Short Message Service (SMS).

One consequence of adopting Spam-I-am is that users can be much more aggressive about their email spam filters, because it prevents legitimate messages that are mistakenly caught by spam filters from going unread.

Spam-I-am's DQM is a useful service in large distributed system where a plentiful, but not infinite, resource needs to be protected from excessive consumption by selfish individuals or applications. Some examples include disk storage in a file or backup system and computational cycles in a peer-to-peer system or computational economy.

Acknowledgments

We thank David Andersen, Susan Hohenberger, David Maziéres, Ronald Rivest, Stuart Schechter, Mythili Vutukuru, Michael Walfish, and the HotNets reviewers for their comments. This work was supported by the NSF under Cooperative Agreement No. ANI-0225660.

8. REFERENCES

[1] ABADI, M., BIRRELL, A., BURROWS, M., DABEK, F., AND WOBBER, T. Bankable postage for network services. In *Proc. Advances in Computing Science* (Mumbai, India, December 2003).

[2] BACK, A. Hashcash. `http://www.cypherspace.org/adam/hashcash/`.

[3] Bonded Sender Program. `http://www.bondedsender.com/info_center.jsp`.

[4] BOUTIN, P. Interview with a spammer. *Infoworld* (April 2004).

[5] Brightmail, inc.: Spam percentages and spam categories. `http://www.brightmail.com/spamstats.html`.

[6] Camram. `http://www.camram.org/`.

[7] CASTRO, M., DRUSCHEL, P., GANESH, A., ROWSTRON, A., AND WALLACH, D. S. Secure routing for structured peer-to-peer overlay networks. In *Proc. 5th USENIX OSDI Conf.* (Boston, Massachusetts, December 2002).

[8] DWORK, C., GOLDBERG, A., AND NAOR, M. On memory-bound functions for fighting spam. In *Proc. CRYPTO Conf.* (2003), pp. 426–444.

[9] DWORK, C., AND NAOR, M. Pricing via processing or combatting junk mail. In *Proc. CRYPTO Conf.* (1992), pp. 139–147.

[10] Electronic Cash. `http://www.tcs.hut.fi/~helger/crypto/link/protocols/ecash.html`.

[11] GOODMAN, J., AND ROUNTHWAITE, R. Stopping Outgoing Spam. In *Proc. ACM Conf. on Electronic Commerce (EC)* (New York, NY, May 2004).

[12] Habeas Sender Warranted Email. `http://www.habeas.com/`.

[13] Project IRIS. `http://project-iris.net/`.

[14] KRISHNAMURTHY, B., AND BLACKMOND, E. SHRED: Spam Harassment Reduction via Economic Disincentives. `http://www.research.att.com/~bala/papers/shred-ext.ps`, 2004.

[15] LAURIE, B., AND CLAYTON, R. "Proof-of-Work" Proves Not to Work. `http://www.apache-ssl.org/proofwork.pdf`, May 2004.

[16] OpenDHT. `http://openhash.org/`.

[17] The Penny Black Project. `http://research.microsoft.com/research/sv/PennyBlack/`.

[18] RAYMOND, P. R. System and method for discouraging communications considered undesirable by recipients. US Patent 6,697,462, February 2004.

[19] SpamAssassin. `http://spamassassin.apache.org/`.

[20] Sender Policy Framework. `http://spf.pobox.com/`.

[21] VON AHN, L., BLUM, M., HOPPER, N. J., AND LANGFORD, J. CAPTCHA: Using Hard AI Problems for Security. In *Proc. EUROCRYPT* (Warsaw, Poland, May 2003).