Zmail : Zero-Sum Free Market Control of Spam

Benjamin J. Kuipers Alex X. Liu Aashin Gautam Mohamed G. Gouda

Department of Computer Sciences
The University of Texas at Austin
Austin, Texas 78712-1188, U.S.A.
{kuipers, alex, aashin, gouda}@cs.utexas.edu

Abstract

The problem of spam is a classic "tragedy of the commons" [13]. We propose the Zmail protocol as a way to preserve email as a "free" common resource for most users, while imposing enough cost on bulk mailers so that market forces will control the volume of spam. A key idea behind Zmail is that the most important resource consumed by email is not the transmission process but the end user's attention. Zmail requires the sender of an email to pay a small amount (called an "e-penny") which is paid directly to the receiver of the email.

Zmail is thus a "zero-sum" email protocol. Users who receive as much email as they send, on average, will neither pay nor profit from email, once they have set up initial balances with their ISPs to buffer the fluctuations. Spammers will incur costs that will moderate their behavior. Importantly, Zmail requires no definition of what is and is not spam, so spammers' efforts to evade such definitions become irrelevant. We describe methods within Zmail for supporting "free" goods such as volunteer mailing lists, and for limiting exploitation by email viruses and zombies.

Zmail is not a micro-payment scheme among end-users. It is an accounting relationship among "compliant ISPs", which reconcile payments to and from their users. Zmail can be implemented on top of the existing SMTP email protocol. We describe an incremental deployment strategy that can start with as few as two compliant ISPs and provides positive feedback for growth as Zmail spreads over the Internet.

1. Overview

1.1. The Problem of Spam

The volume of unsolicited commercial email – spam – has grown to the point where it is not only a nuisance to individual users, but it imposes a major load on ISPs and threatens the social viability of the Internet itself. The problem of spam is a classic "tragedy of the commons" [13]. The Internet provides email (as well as other services) as a "free" common resource shared by all users. Spam is an abuse of this common resource for private profit, enabled by the extremely low cost of sending large amounts of email. For email users, spam costs significant time and effort to separate legitimate email from spam, during which important personal or business email could be accidentally lost. For ISPs, spam costs a great deal of bandwidth, storage, and processing time.

In the last few years, spam has grown dramatically. According to Brightmail [5], more than 60% of all email traffic in April 2004 was spam as compared to only 8% in 2001. In 2003, the cost of deploying additional email processing servers due to the influx of spam was $10 billion in the United States according to Ferris Research and $20.5 billion worldwide according to Radicati Group [25]. These two figures do not include losses of worker productivity caused by spam. Gartner Group has estimated that on average, a business with 1,000 employees loses $300,000 a year in worker productivity due to spam [19]. Spam has become a problem of national importance and solving this problem has become an urgent need.

A variety of legislative, filtering, and economic approaches to the spam problem have been proposed. (In Section 2, we review these approaches in detail.) However, previous approaches could not solve the spam problem effectively.

1.2. A Zero-Sum Free Market Solution

In this paper, we propose the Zmail protocol as a way to preserve email as a "free" common resource for most users, while imposing enough cost on bulk mailers so that market forces will control the volume of spam. By reversing the economics of the current Internet email system, spammers lose their free ride. A key idea behind Zmail is that the most important resource consumed by email is not the transmission process but the recipient's attention. Zmail requires the sender of an email to pay a small amount of money directly to the receiver of the email. The money earned by a user can be used to send email or can be exchanged for real money. The cost of sending (or value of receiving) one email message is a unit called an *e-penny*. For simplicity, assume that the "real money" cost of one e-penny is $0.01.

Zmail is thus a "zero-sum" email protocol because any complete transaction in Zmail is zero-sum: the amount of money charged to a sender is equal to the amount of money rewarded to the corresponding receiver. Unlike the postal system in real life where the money that a sender paid goes to the postal service providers to pay for physical message transmission, in Zmail, the money that a sender paid goes to the receiver, while the role of ISPs is to facilitate email transfer and payment management.

Next we discuss the impact of market forces on the three relevant players: spammers, normal email users and ISPs.

1. Spammers: The cost of sending spam will increase by at least two orders of magnitude, significantly changing the economics of sending huge numbers of unsolicited email messages. The response rate required to break even will increase similarly. Bulk email advertising will continue to exist, but the incentives will favor more targeted advertising to populations of readers who are likely to be interested in the product. The amount of spam will undoubtedly decrease substantially.

2. Normal Users: Users who receive as much email as they send, on average, will neither pay nor profit from email, once they have set up initial balances with their ISPs to buffer the fluctuations. Typically a normal user does not need to pay real money to buy e-pennies for sending email. Even a normal user who needs to send more email than she receives and who is unwilling to spend money can easily solve this problem by subscribing to some commercial email services in which she is interested. When a normal user receives spam accidentally, it can be viewed as a windfall rather than a nuisance because of the payment received.

3. ISPs: The Zmail protocol significantly reduces spam and therefore reduces the overhead costs of ISPs by saving their disk space, bandwidth, and computational cost for running spam filters.

One important property of Zmail is that it requires no definition of what is and is not spam. One great difficulty experienced by existing anti-spam approaches is that it is almost impossible to define what spam is. One person's annoying spam may be another person's useful information. Spammers can often find ways to bypass existing spam filters by techniques such as deliberate misspelling. False positives in filtering out spam are not acceptable because modern life depends so heavily upon email communication and because huge losses could result from incorrectly discarded email. Using Zmail, spammers' efforts to evade definitions of spam become irrelevant.

Zmail has many other nice features, such as supporting mailing lists and limiting the exploitation by email virus and zombies, which we discuss in Section 6.

1.3. The Zmail Protocol

Zmail can be implemented on top of the current Internet email protocol SMTP (Simple Mail Transfer Protocol) [24]. Zmail requires no change to SMTP. In the Zmail protocol, all the payments are handled automatically and the underlying economics remains almost transparent to the users. Note that Zmail is not a micro-payment scheme among end-users. It is an accounting relationship among "compliant ISPs", which reconcile payments to and from their users. The actions that a user needs to take to send or receive an email remain the same as those of the current Internet email system. Normal users will hardly find any difference from current email systems.

Zmail can be deployed incrementally. It can be bootstrapped with as few as two compliant ISPs. People will not experience disruption in using email services. The good experience of the users of compliant ISPs will attract more people to switch to compliant ISPs and more ISPs will therefore become compliant. Eventually, we envision that Zmail will spread over the Internet.

The rest of this paper proceeds as follows. In Section 2, we review and examine existing or proposed approaches to the problem of spam. We present a brief introduction to the Abstract Protocol notation in Section 3, while in Section 4, we describe the details of the Zmail protocol using this notation. In Section 5, we discuss several related issues of the Zmail scheme. Section

6 concludes. The formal specification of the Zmail protocol is given in the appendix.

2. Related Work

The various approaches that have been developed or proposed to cut down spam fall roughly into the following three categories: legal approaches, filtering approaches, and economic approaches. Next, we review these three approaches.

2.1. Legal Approaches

Legal approaches are intended to regulate email communications. Many U.S. states such as Texas, Virginia and Washington have passed anti-spam laws. For example, a Texas anti-spam law enacted in June 2003 requiring that unsolicited commercial email messages include a label ("ADV:" or "ADV: ADULT ADVERTISEMENT") at the beginning of the subject line, and include a functioning return email address for recipients to request removal from future mailings which must be honored. This law also prohibits unsolicited commercial email messages with falsified routing information. False, deceptive, or misleading subject lines are prohibited in all commercial email messages, as is the unauthorized use of a third party's domain name. The federal anti-spam law, the Controlling the Assault of Non-Solicited Pornography and Marketing Act (CAN-SPAM Act), was recently passed in December of 2003. This act also let the Federal Trade Commission (FTC) to establish a national "do-not-email" registry of addresses of persons and entities who do not wish to receive spam. The FTC would be empowered to impose civil penalties upon those who send spam to addresses listed on the registry. A list of anti-spam laws together with their summaries is available at [27].

There are two major disadvantages of legal approaches. First, it is extremely hard to define precisely what kind of email should be prohibited. If an anti-spam law is written narrowly, then it does not have much effect on stopping spam. Ironically, some existing anti-spam laws in fact legalize some forms of spam. If an anti-spam law is written widely, then it could intrude upon the First Amendment right to free speech. One person's annoying spam may be another person's welcome source of information.

Second, even assuming that a perfect anti-spam law exists, enforcement of this law is difficult because spammers can simply move their operations to a country that has no anti-spam laws. In fact, a lot of spammers have already done so. According to the latest report published by Sophos in August 2004 on the countries from which most spam messages originate, 57.47% spam are originated from outside U.S [8]. Due to the fact that the spam problem has been internationalized and the difficulties of jurisdiction, the national "do-not-email" registry can not effectively stop spam. The recent report [23] that was made by FTC to Congress in June 2004 concluded that a National Do Not Email Registry would fail to reduce the amount of spam consumers receive, might increase it, and could not be enforced effectively.

2.2. Filtering Approaches

Filtering approaches are used to filter out spam after an ISP receives an email and before the ISP delivers the email to its intended receiver. Based on the criteria used to filter out spam, we categorize filtering approaches into *header based filtering approaches* and *content based filtering approaches*.

Header based filtering approaches try to filter out spam according to the header information of email. There are two major types of header based filtering approaches: *blacklist approaches* and *whitelist approaches*. A blacklist is a set of IP addresses that are "known" to be used by spammers. There are some publicly available blacklists such as MAPS Realtime Blackhole List [21], SpamCop Blocking List [29] and SPEWS [30]. A spam filter that uses a blacklist discards any email from an IP address on the blacklist. In contrast, a whitelist is a set of email addresses that are "known" to be non-spammers. A spam filter that uses a whitelist accepts any email from an email address on the whitelist.

Content based filtering approaches try to filter out spam according to email content. Content based spam filters use heuristic or statistical information learned from a collection of spam and legitimate email. Certain research has been done on how to filter out spam based on email content (see [9, 26]). A number of content based spam filters have been developed and deployed, such as SpamAssassin [28], Brightmail [5] and Cloudmark [7]. Note that header based filtering approaches and content based filtering approaches can be used together. For example, an email that passes a whitelist check could be delivered to its intended receiver directly and an email that does not pass a whitelist checking could be sent to a content based spam filter for further examination.

There are two major disadvantages of filtering approaches. First, spam filters are vulnerable to false positive errors. A false positive is when a legitimate email is mistaken as spam and wrongly discarded by a spam filter. Newsletters and paid subscriptions have a high probability of being classified as spam. A false positive error could possibly be a disaster for a user. For example, if a job offer email with "Great Offer" as its subject

title is discarded by a spam filter, the intended receiver of the email could lose a good job opportunity. According to Jupiter Research [15], the cost of discarding such legitimate email is expected to soar to $419 million in 2008 from $230 million in 2003, although the percentage of wrongly blocked legitimate email will drop from 17% today to just under 10% in 2008.

Second, spammers can foil spam filters. To combat blacklists, spammers can use well-known ISPs or some hacked computers to send spam. To take advantage of whitelists, spammers usually forge their domain names. To deceive content based spam filters, spammers have begun to use tricks. For example, spammers may deliberately misspell sensitive words, such as spell "sex" as "se><". Sometimes spammers embed text information into a picture. No matter how smart a spam filter program is, spammers can always find ways to deceive it.

2.3. Economic Approaches

Economic approaches fight spam by requiring senders pay for their email. According to what senders pay, we categorize economic approaches into three types: *human effort based approaches*, *computational cost based approaches*, and *monetary value based approaches*.

In human effort based approaches, what senders pay is human effort. This type of approach works in a challenge-response fashion: when an ISP receives an email, it first holds the email and sends back a challenge. A challenge, for example, can be an image that contains numbers and letters; the sender is then required to send back the text content of the image. Visually processing such an image is trivial for a human but hard for a computer. If the response from the sender is correct, then the email will be delivered to its intended receiver; otherwise the email will be discarded. This human action based approach has been adopted by some commercial email products such as Mailblocks [18] and Active Spam Killer [3]. The major disadvantage of this approach is that it is inconvenient, inefficient and sometimes a challenge can be perceived as rude by the sender.

In computational cost based approaches, what senders pay is computer processing time. This type of approach works also in a similar challenge-response fashion. The difference here is that a challenge is a computer solvable problem. Such a problem must be CPU or memory intensive enough to prevent a spammer from sending out a huge amount of email in a short period. For example, one challenge might need one minute to compute, which therefore limits the sending rate of a spammer to no more than one email per minute. Some research has been done on computational based economic approaches (see [1, 2, 4, 6, 10, 11] and the Microsoft Penny Black Project [22] for details). There are two major disadvantages of this approach. First, email systems become significantly inefficient in sending and receiving email. Second, the cost to ISPs for sending out email is dramatically increased, which makes ISPs extremely reluctant to use this type of approach.

In monetary value based approaches, what senders pay is a certain value of money. Existing monetary value based approaches mainly include the SHRED scheme [16] and the Vanquish scheme [31]. In SHRED and Vanquish, the receiver of an email can trigger a payment from the sender of the email to the sender's ISP if the receiver finds the email unwanted. There are four major disadvantages of SHRED and Vanquish. First, SHRED and Vanquish not only fail to reduce human effort on separating legitimate email from spam, but actually increase it. For each spam received by a user, instead of taking one action to delete the email, she needs to take an extra action to trigger the payment associated with the email. Second, the receiver of a spam may not be motivated to spend this extra effort because she is not directly rewarded for doing so. Note that if the receiver of a spam indeed triggers the payment associated with the payment, it is the sender's ISP, not the receiver, who gets the payment. Third, a spammer can collude with its ISP and avoid any monetary cost. Last, the storage and computational cost for an ISP to collect an individual payment could possibly exceed the monetary value of the payment because the payment for each spam is handled individually in both SHRED and Vanquish, and the monetary value of the payment is usually small, only one penny or even a fraction of a penny.

Zmail also falls into this category. However, Zmail overcomes the above four weaknesses of SHRED and Vanquish. First, in Zmail, the receiver of a spam does not need to take any action for the sender of the email to be charged. All payments are handled automatically without user's intervention or even notice. Second, in our approach, a spam would be viewed as a windfall rather than a nuisance by its receiver because the receiver is paid for receiving the email. Third, in our approach, if an ISP colludes with spammers, the ISP can be discovered. Last, in our approach payments are handled in a bulk fashion; therefore, the cost of handling payments is small.

3. Abstract Protocol Notation

Here we give a brief introduction to the Abstract Protocol notation [12]. In this notation, each process in a protocol is defined by sets of constants, variables, parameters, and actions. For instance, in a protocol consisting of two processes p and q and two opposite-direction channels, one from p to q and one from q to p, process p can be defined as follows:

process p
const ⟨name of constant⟩ : ⟨type of constant⟩
  ...
inp ⟨name of input⟩ : ⟨type of input⟩
  ...
var ⟨name of variable⟩ : ⟨type of variable⟩
  ...
par ⟨name of parameter⟩ : ⟨type of parameter⟩
  ...
begin
  ⟨action⟩
□ ⟨action⟩
□ ...
□ ⟨action⟩
end

The constants of process p have fixed values. Inputs of process p can be read, but not updated, by the actions of process p. Variables of process p can be both read and updated by the actions of process p. Comments can be added anywhere in process p; every comment is placed between the two brackets { and }.

Each ⟨action⟩ of process p is of the form:

$$\langle guard \rangle \ \rightarrow \ \langle statement \rangle$$

The guard of an action of process p is of one of the following three forms: (1) a boolean expression over the constants and variables of p, (2) a receive guard of the form "**rcv** ⟨*message*⟩ **from** q", (3) a timeout guard that contains a boolean expression over the constants and variables of every process and the contents of all channels in the protocol. A parameter declared in a process is used to write a finite set of actions as one action, with one action for each possible value of the parameter.

Executing an action consists of executing the statement of the action. Executing the actions of different processes in a protocol proceeds according to the following three rules. First, an action is executed only when its guard is true. Second, the actions in a protocol are executed one at a time. Third, an action whose guard is continuously true is eventually executed.

The ⟨*statement*⟩ of an action of process p is a sequence of ⟨*skip*⟩, ⟨*send*⟩, ⟨*assignment*⟩, ⟨*selection*⟩, or ⟨*iteration*⟩ statements of the following forms:

⟨*skip*⟩ : **skip**
⟨*send*⟩ : **send** ⟨*message*⟩ **to** q
⟨*assignment*⟩ : ⟨*variable in p*⟩ := ⟨*expression*⟩
⟨*selection*⟩ : **if** ⟨*boolean expression*⟩ → ⟨*statement*⟩
  ...
  □ ⟨*boolean expression*⟩ → ⟨*statement*⟩
  fi
⟨*iteration*⟩ : **do** ⟨*boolean expression*⟩ → ⟨*statement*⟩
  od

There are two channels between the two processes: one is from p to q, and the other is from q to p. Each message sent from p to q remains in the channel from p to q until it is eventually received by process q. Messages that reside simultaneously in a channel form a sequence and are received, one at a time, in the same order in which they were sent.

4. Specification of Zmail Protocol

In this section, we describe the details of the Zmail protocol using the Abstract Protocol notation. In the Zmail protocol, there are two types of parties: ISPs and banks. The major role of ISPs is to send and receive email for their users. We do not assume that every ISP has to join the Zmail protocol, although we expect that eventually most ISPs will. We call the ISPs that are running the Zmail protocol "*compliant ISPs*". The major role of the banks is to manage e-pennies: exchange real money for e-pennies and exchange e-pennies for real money. For simplicity, we assume there is only one central bank. Every compliant ISP is registered with the bank and has an account in the bank. Instead of having the bank itself manage e-pennies for all individual email users, which is inefficient, we let the bank manage e-pennies for each compliant ISP and let each compliant ISP manage e-pennies for its own users.

The constants, inputs, variables and parameters in each ISP process are defined as follows:

process *isp*[i:0..n-1]

const n : **integer**, {# of ISPs}
  m: **integer**, {# of users per ISP}
  compliant : **array** [0..n-1] **of boolean**
  {array compliant indicates which ISP is compliant}

inp B_b: **integer**, {public key of bank}
  limit: **array** [0..n-1] **of integer**,
    {*limit*[*j*]: max # of emails sent per day for user j}
  maxavail, *minavail* : **integer**,
    {*maxavail*, *minavail* are two thresholds for *avail*}

var *avail* : **integer**, {# of e-pennies available for users to buy}
  account: **array** [0..m-1] **of integer**, {balance of real pennies}
  balance : **array** [0..m-1] **of integer**, {balance of e-pennies}
  sent : **array** [0..m-1] **of integer**, {# of emails sent}
  credit : **array** [0..n-1] **of integer**, {sending&receiving record}
  cansend, *canbuy*, *cansell* : **boolean**, {initial value: true}
  buyvalue, *sellvalue* : *minavail*..*maxavail*,

```
        accepted : boolean,
        seq, seq' : integer, {initial value: 0}
        ns1, nr1, ns2, nr2 : integer, {nonces}
        x          : integer,
        s, r       : 0..m-1,
        j          : 0..n-1

par     g : 0..n-1
        t : 0..m-1
```

Note that each ISP process has three constants n, m, and *compliant*, and all ISP processes share the same value for each of the three constants. Constant n is the number of ISPs. The n ISP processes are $isp[0], isp[1], \cdots, isp[n-1]$. For simplicity, we assume each ISP has the same number of users, and constant m is this number. Constant "*compliant*" is a boolean array of length n, and it indicates which ISP is compliant. This array is maintained and published by the bank. When an ISP $isp[j]$ changes its status from non-compliant to compliant, the bank flips $compliant[j]$ from false to true, and broadcast this new "*compliant*" array to every compliant ISP. For simplicity, in this paper, we assume that the "*compliant*" array does not change its value.

The constants, inputs, variables, and parameters in the bank process are defined as follows:

```
process bank

const n : integer, {# of ISPs}
        compliant : array [0..n-1] of boolean
        {array compliant indicates which ISP is compliant}

inp     B_b : integer, {public key of bank}
        R_b : integer  {private key of bank}

var     account : array [0..n-1] of integer,
                  {balance of real pennies for every ISP}
        verify  : array [0..n-1] of array [0..n-1] of integer,
                  {initial value: 0}
        seq     : integer, {initial value: 0}
        total   : integer, {initial value: 0}
        i, j    : integer, {initial value: 0}
        credit  : array [0..n-1] of integer,
        x, y    : integer,
        nr      : integer, {nonce}
        canrequest : boolean, {initial value: 0}

par     g : 0..n-1
```

Next, we discuss the details of the compliant ISP $isp[i]$ and the bank, during which we will explain the meaning of every input and variable of ISPs and the bank.

4.1. Zero-sum email transfer

In process $isp[i]$, each user specifies the maximum number of emails that they can send out to compliant ISPs in the array *limit*. The purpose of setting this limit is to mitigate the potential cost incurred by email viruses. Detail discussion of email viruses is in Section 5. Process $isp[i]$ uses the array *sent* to keep track of the number of emails that each user sends to compliant ISPs in each day. At the end of every day, array *sent* is reset to 0. Process $isp[i]$ maintains the balance of e-pennies for every user using the array *balance*. Process $isp[i]$ records the transaction of e-penny exchanges with other ISPs in the array *credit*. The initial value of every element in array *credit* is zero.

Process $isp[i]$ maintains a variable called "*cansend*". When *cansend* is true, process $isp[i]$ can send out email. When process $isp[i]$ sends an email to a compliant ISP $isp[j]$, $credit[j]$ is increased by one; when process $isp[i]$ receives an email from a compliant ISP $isp[j]$, $credit[j]$ is reduced by one. Later we will show that the bank can detect misbehaved ISPs using the information in the *credit* array of every ISP. The initial value of *cansend* is true. The pseudocode of the process $isp[i]$ for sending email is as follows. Note that the keyword **any** means an arbitrary value from its domain of values, which is used to simulate a user's input.

```
□ cansend →
    s := any; j := any; r := any;
    {user s of isp[i] wants to send email to user r of isp[j]}
    if i = j → if balance[s] ≥ 1 ∧ sent[s] < limit[s] →
                                  balance[s]:= balance[s] − 1;
                                  balance[r]:= balance[r] + 1;
                                  sent[s]    := sent[s] + 1;
                                  {deliver email(s, r) to user r}
               □ balance[s] = 0 ∨ sent[s] ≥ limit[s] → skip
               fi
    □ i ≠ j → if compliant[j] →
                 if balance[s] ≥ 1 ∧ sent[s] < limit[s] →
                                  balance[s]:= balance[s] − 1;
                                  credit[j]  := credit[j] + 1;
                                  sent[s]    := sent[s] + 1;
                                  send email(s, r) to isp[j]
                 □ balance[s] = 0 ∨ sent[s] ≥ limit[s] → skip
                 fi
               □ ∼ compliant[j] → send email(s, r) to isp[j]
               fi
    fi
```

The pseudocode for receiving email is as follows:

```
□ rcv email(s, r) from isp[g] →
    if   compliant[j] → balance[r] := balance[r] + 1;
                        credit[g] := credit[g] − 1
                        {deliver the email to r}
    □ ∼ compliant[g] → skip {deliver to r or discard it}
    fi
```

The pseudocode for resetting array *sent* to 0 at the end of every day is as follows:

```
□ true→ {execute at the end of every day}
      x := 0; do x < n → sent[x] := 0 od
```

4.2. Transaction With Users

Process $isp[i]$ maintains a pool of e-pennies that its users can buy. The amount of e-pennies in this pool is stored in a variable named *avail*. Each user has an account of real money with their ISP, and process $isp[i]$

maintains the balance of real pennies for every user using array *account*. A user can buy and sell e-pennies with their ISP. The pseudocode of the process *isp*[*i*] for managing transactions with users is as follows:

```
□ account[t] > 0 → x := any; {user t wants to buy x e-pennies}
    if account[t] ≥ x ∧ avail ≥ x → account[t]:= account[t] − x;
                                    balance[t] := balance[t] + x;
                                    avail[t]   := avail[t] − x;
    □ account[t] < x ∨ avail < x →skip
    fi
□ balance[t] > 0 → x := any; {user t wants to sell x e-pennies}
    if balance[t] ≥ x → account[t]:= account[t] + x;
                        balance[t] := balance[t] − x;
                        avail[t]   := avail[t] + x;
    □ balance[t] < x →skip
    fi
```

4.3. Transaction With Bank

For the variable *avail*, process *isp*[*i*] specifies one lower bound named *minavail* and one upper bound named *maxavail*. When *avail* < *minavail*, process *isp*[*i*] needs to buy some e-pennies from the bank; when *avail* > *maxavail*, process *isp*[*i*] needs to sell some e-pennies back to the bank. In the communication between the bank and the process *isp*[*i*] for buying and selling e-pennies, we add nonces to prevent message replay attacks. A nonce is an integer generated by a function called NNC. The sequence of nonces generated by a process using the function NNC has two properties: unpredictability and nonrepetition. The pseudocode of the process *isp*[*i*] for managing transactions with the bank is as follows. Note that $DCR(B_b, x)$ denotes the result of decrypting x using the key B_b, $NCR(k, d)$ denotes the encryption of data item d using key k, and $DCR(k, d)$ denotes the decryption of data item d using key k.

```
□ canbuy→
        if avail < minavail →
            canbuy := false; buyvalue := any; ns1 := NNC;
            send buy(NCR(B_b, (buyvalue|ns1)) to bank;
        □ avail ≥ minavail → skip
        fi

□ rcv buyreply(x) from bank →
        nr1, accepted := DCR(B_b, x);
        if ns1 = nr1 → canbuy := true
                       if accepted → avail := avail + buyvalue
                       □ ∼ accepted → skip
                       fi
        □ ns1 ≠ nr1 → skip
        fi

□ cansell →
        if avail > maxavail →
            cansell := false; sellvalue := any; ns2 := NNC;
            send sell(NCR(B_b, (sellvalue|ns2)) to bank;
        □ avail ≤ maxavail → skip
        fi

□ rcv sellreply(x) from bank →
        nr2 := DCR(B_b, x);
        if ns2 = nr2 → avail := avail − sellvalue; cansell := true
        □ ns2 ≠ nr2 → skip
        fi
```

Every compliant ISP has an account of real money with the bank, and the bank stores the balance of real pennies of compliant ISPs in its array *account*. The pseudocode of the process *bank* for managing transactions with compliant ISPs is as follows:

```
□ rcv buy(x) from isp[g] →
    nr, y := DCR(R_b, x); {isp[g] wants to buy y e-pennies}
    if account[g] ≥ y → account[g] := account[g] − y;
                        send buyreply(NCR(R_b, nr|true)) to isp[g];
    □ account[g] < y → send buyreply(NCR(R_b, nr|false)) to isp[g];
    fi

□ rcv sell(x) from isp[g] →
    nr, y := DCR(R_b, x); {isp[g] wants to sell y e-pennies}
    account[g] := account[g] + y;
    send sellreply(NCR(R_b, nr)) to isp[g];
```

4.4. Detecting Misbehavior

We have seen the operations on the *credit* array in both the sender's and receiver's end. Note that in a certain time period that all the email sent from *isp*[*i*] to *isp*[*j*] and all the email from *isp*[*j*] to *isp*[*i*] are received by these two ISPs, the value of *credit*[*j*] in process *isp*[*i*] plus the value of *credit*[*i*] in process *isp*[*j*] should be zero. Otherwise, at least one of the two ISPs has misbehaved. The bank needs to gather the *credit* array from every ISP periodically, say one time a month, and then detect the suspected misbehaved ISPs, based on which the bank may make further investigation. Because each compliant ISP has been authenticated to be "good guys", we expect the chance of inconsistency in *credit* arrays is extremely small.

To take a snapshot of *credit* arrays of all compliant ISPs, we use a simple timeout method. When the bank wants to gather *credit* arrays, it sends out a request message to every compliant ISP. When a compliant ISP receives the request message, it stops sending out any email for a certain time period, say 10 minutes, to ensure that every email that it sent out is received. After this time period, the ISP sends its *credit* array to the bank. Thereafter, the ISP reset its *credit* array to zero because a new billing period starts. Each request message from the bank has a sequence number, which is used to prevent message reply attacks. The pseudocode of process *isp*[*i*] for receiving request message from the bank and sending the *credit* array to the bank is as follows.

```
□ rcv request(x) from bank →
    seq′ := DCR(B_b, x);
    if seq = seq′ → cansend := false; timeout after 10 minutes
    □ seq ≠ seq′ → skip
    fi
```

```
□ timeout expired →
      send reply(NCR(B_b, credit)) to bank;
      x := 0; do x < n → credit[x] := 0; x := x + 1 od;
      cansend := true;
      seq := seq + 1
```

Note that the 10 minutes timeout period is only experienced by ISPs, not email users. An email user still can instruct their ISP to send emails during the timeout period, although these emails will be buffered and sent right after the timeout expires. Here we choose this timeout method for the simplicity of discussion. In implementing Zmail, one could choose other methods to take a snapshot of the *credit* arrays of all compliant ISPs.

Every time the bank wants to verify the compliance of ISPs, it first sends request to every compliant ISP. After the bank receives the *credit* array from every compliant ISP, the bank starts to verify that for every two compliant ISPs $isp[i]$ and $isp[j]$, the value of $credit[j]$ in process $isp[i]$ plus the value of $credit[i]$ in process $isp[j]$ should be zero. The frequency of this consistency checking may be once a week or once a month, for example. The pseudocode of the process *bank* for consistency checking is as follows:

```
□ canrequest →
    i := 0;  total := 0;
    do i < n → if   compliant[i] →
                      total := total + 1;
                      send request(NCR(R_b, seq)) to isp[i];
                 □ ∼compliant[i] → skip
                 fi;
                 i := i + 1
    od;
    canrequest := false

□   rcv reply(x) from isp[g] →
    if   compliant[g]→
           credit := DCR(R_b, x);  total := total − 1;  i := 0;
           do i < n → verify[i, g] := credit[i]; i := i + 1 od;
    □ ∼compliant[g]→ skip
    fi

□   total = 0∧ ∼ canrequest →
    i := 0;  j := 0;
    do i < n →
       j := 0
       do j < n →
          if verify[i, j] + verify[j, i] = 0 → skip
          □ verify[i, j] + verify[j, i] ≠ 0 → skip {report error}
          fi;
          verify[i, j] := 0;  j := j + 1
       od;
       i := i + 1
    od;
    canrequest := true
```

5. Discussion

In this section, we discuss the issues of mailing lists, email viruses, incremental deployment and bank setup.

Mailing Lists. A mailing list works as follows. Each mailing list has a list server that runs a mailing list server program. Two of the most popular mailing list server programs are Listserv [17] and Majordomo [20]. A list server consists of a subscriber database and an email distributor. The subscriber database consists of all the email addresses of the people who have joined the list. Each time a subscriber wants to send an email to everyone in the mailing list, she sends one email to the email address of the email distributor. Each time the email distributor receives an email from one of its subscribers, it will forward the same email to every email address in the subscriber database. In moderated mailing list, the email distributor could be a human, while in unmoderated mailing list, the email distributor is usually a program.

Directly applying the economic model of the Zmail protocol to mailing lists may impose too much cost to the distributor because every time that it receives an email from a subscriber, it needs to send out a huge number of emails. To compensate the cost of the distributor, we define a special-purpose email message that would be automatically generated by the receiver's ISP or email client and sent back to the email distributor, acknowledging the receipt of the mailing-list message. This acknowledgment email returns the e-penny back to the distributor. The difference between acknowledgment email and normal email is that acknowledgment email can be processed automatically, rather than being delivered to the receiver's inbox for human attention.

An additional benefit of this automated acknowledgement mechanism is that the email distributor can automatically keep track of which addresses do not acknowledge messages and should be removed from its subscriber database. Therefore, the email distributor can keep its subscriber database clean and up-to-date.

Zombies and Email Viruses. A virus can allow a user's PC to be exploited without the user's consent or even knowledge. An email virus may send messages to the user's entire address book. If a virus has made the user's PC into a "zombie", it could be used to send out large amounts of spam at the user's expense.

To limit this, and more importantly to allow the detection of "zombified" PCs, ISPs can enforce a user specified limit on the number of e-pennies the user is willing to spend per day. (Recall the *limit* array in our formal specification of Zmail in Section 4). Exceeding this limit blocks further outgoing mail (for that day), and the user is sent a warning message to check for viruses. In addition to limiting the user's liability for the e-penny cost of virus-sent email, this provides a new mechanism for detecting, limiting, and disinfect-

ing "zombie" PCs once they become active.

Incremental Deployment. One feature of the Zmail protocol is that it can be deployed incrementally, starting with two compliant ISPs. We have seen that in the Zmail protocol a non-compliant ISP can still send email to a compliant ISP, but a user in a compliant ISP may decide to segregate or discard email from non-compliant ISPs, or require any email from a non-compliant ISP to pass a spam filter. As more and more ISPs become compliant, more people would choose not to accept any email from a non-compliant ISP, which in turn causes more people to use compliant ISPs and more ISPs to become compliant.

Bank Setup. In the Zmail protocol, we assume that there is a central bank. A central authority is not difficult to set up on the Internet. In fact, the Internet already has some central authorities such as IANA [14] that controls the assignment of IP addresses. In fact, the role of the bank in the Zmail protocol can be implemented as a set of distributed banks or a hierarchy of banks. It is fairly straightforward to extend the Zmail protocol to incorporate multiple collaborating banks.

6. Conclusions

In the current Internet, spam traffic has exceeded the traffic of legitimate email. Solving the spam problem has become an urgent need due to the huge financial losses caused by spam. The root of the spam problem is that the current email system of the Internet provides free ride to spammers. In this paper, we propose the Zmail protocol that stops the free ride for spammers and therefore solves the spam problem fundamentally, while preserving the essentially "free" nature of email for normal users. We provide a formal specification of the Zmail protocol using the Abstract Protocol notation.

References

[1] M. Abadi, A. D. Birrell, M. Burrows, F. Dabek, and T. Wobber. Bankable postage for network services. In *Proc. of the 8th Asian Computing Science Conference*, December 2003.

[2] M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately hard, memory-bound functions. In *Proc. of the 10th Annual Network and Distributed System Security Symposium*, February 2003.

[3] Active Spam Killer, http://www.paganini.net/ask/. 2004.

[4] Adam Back. Hashcash - A Denial of Service Counter-Measure. Available at http://www.hashcash.org/papers/hashcash.pdf. 2002.

[5] Brightmail, http://www.brightmail.com/. 2004.

[6] CAMRAM, http://www.camram.org/. 2004.

[7] Cloudmark, http://www.cloudmark.com/. 2004.

[8] Dirty dozen spam producing countries, http://sophos.com/spaminfo/articles/dirtydozenaug04.html. 2004.

[9] H. Drucker, D. Wu, and V. N. Vapnik. Support vector machines for spam categorization. *IEEE Transactions on Neural Networks*, 10(5):1048–1054, 1999.

[10] C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam. In *Crypto 2003*, 2003.

[11] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Proc. of the 12th Annual International Cryptology Conference on Advances in Cryptology (Crypto-92), Springer-Verlag Lecture Notes in Computer Science*, volume 740, pages 139–147, 1992.

[12] M. G. Gouda. *Elements of Network Protocol Design*. John Wiley & Sons, New York, New York, 1th edition, 1998.

[13] G. Hardin. The tragedy of the commons. *Science*, 162(1968):1243–1248.

[14] IANA, http://www.iana.org/. 2004.

[15] Jupiter Research, http://www.jupiterresearch.com/. 2004.

[16] B. Krishnamurthy and E. Blackmond. Shred: Spam harassment reduction via economic disincentives. *Unpublished manuscript*, 2004.

[17] Listserv, http://www.lsoft.com/. 2004.

[18] Mailblocks, http://about.mailblocks.com/. 2004.

[19] MailWatch, http://www.easylink.com/services_north_america/boundary_spam.cfm. 2004.

[20] Majordomo, http://www.greatcircle.com/majordomo/. 2004.

[21] MAPS RBL, http://www.mail-abuse.com/services/mds_rbl.html. 2004.

[22] Microsoft Penny Black Project, http://research.microsoft.com/research/sv/PennyBlack/. 2004.

[23] National Do Not Email Registry: A Report to Congress, http://www.ftc.gov/reports/dneregistry/report.pdf. 2004.

[24] J. B. Postel. Simple mail transfer protocol. http://www.faqs.org/rfcs/rfc821.html. August 1982.

[25] S. Hansell. Diverging estimates of the costs of spam. *New York Times*, August 28, 2003.

[26] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A bayesian approach to filtering junk E-mail. In *Proc. of AAAI Workshop on Learning for Text Categorization*, Madison, Wisconsin, 1998.

[27] Spam Laws, http://www.spamlaws.com/. 2004.

[28] SpamAssassin, http://spamassassin.apache.org/. 2004.

[29] SpamCop Blocking List, http://www.spamcop.net/bl.shtml. 2004.

[30] SPEWS, http://www.spews.org/filter.html. 2004.

[31] Vanquish AntiSpam Appliance, http://www.vanquish.com/products/pr_vqsa.shtml. 2004.