Zmail : Zero-Sum Free Market Control of Spam

Benjamin J. Kuipers, Alex X. Liu[1], Aashin Gautam[2], 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" [10]. 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 ESPs (Email Service Providers) 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 ESPs", 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 ESPs and provides positive feedback for growth as Zmail spreads over the Internet.

[1] Alex X. Liu is the corresponding author of this paper.

[2] Aashin Gautam is currently with National Instruments. He participated in this work while he was an undergraduate student in The University of Texas at Austin.

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 Email Service Providers (ESPs), such as Hotmail (http://www.hotmail.com), and threatens the social viability of the Internet itself. The problem of spam is a classic "tragedy of the commons" [10]. 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 ESPs, spam costs a great deal of bandwidth, storage, and processing time.

In the last few years, spam has grown dramatically. According to Brightmail (http://www.brightmail.com/), 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 [14]. Spam has become a problem of national importance and solving this problem has become an urgent need.

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 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 ESPs 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 ESPs.

1. Spammers: The cost of sending spam will increase by at least two orders of magnitude, significantly changing the economics of sending amount of unsolicited email. 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 ESPs 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. ESPs: The Zmail protocol significantly reduces spam and therefore reduces the overhead costs of ESPs 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) [13]. 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 ESPs", 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 ESPs. People will not experience disruption in using email services. The good experience of the users of compliant ESPs will attract more people to switch to compliant ESPs and more ESPs 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.

2 Related Work

A variety of legal, filtering, and economic approaches to the spam problem have been proposed. Legal approaches deal with spam by punishing spammers using anti-spam laws. This type of approach has two major drawbacks. First, it is hard to define precisely what kind of email should be prohibited considering the First Amendment right to free speech. Second, it is hard to enforce anti-spam laws because many spammers have moved their operations to countries that have no anti-spam laws and lack of international legal cooperation.

Filtering approaches deal with spam by filtering out spam from incoming email based on text contents. Considerable amount of research has been conducted on this type

of approach (see [3–5, 8, 11, 15, 16]). However, filtering approaches still suffer from two serious drawbacks. First, a spam filter may incorrectly classify a legitimate email as spam, which could cause significant losses. Second, spam filters can be foiled by intelligent, adaptive spammers. For example, spammers may embed text information into an image.

Previous economic approaches fight spam by requiring senders to pay either human efforts or computational costs. They work in a challenge-response fashion: when an ESP receives an email, it first holds the email and sends back a challenge. To solve the challenge, the sender needs to spend either some human efforts (such as visually recognizing the text content of an image), or some computational costs (such as doing some CPU or memory intensive computations). The economic approaches that use human efforts have been adopted by some commercial email products such as Mailblocks (http://about.mailblocks.com/) and Active Spam Killer (http://www.paganini.net/ask/). The major drawback of this type of approach is that it is inconvenient, inefficient and sometimes a challenge can be perceived as rude by the sender. The economic approaches that use computational cost have been extensively studied (see [1, 2, 6, 7, 12]). The major drawback of this type of approach is that it dramatically increases the computational cost incurred on ESPs, which in turn makes ESPs reluctant to deploy this type of approach.

Zmail falls into the category of economic approaches. However, Zmail overcomes the above mentioned weaknesses. In Zmail, the email senders are not required to pay human efforts or computational costs. The efficiency of the email systems is not affected. In addition, a spam message would be viewed as a windfall rather than a nuisance by its receiver because the receiver is paid for receiving it.

3 Abstract Protocol Notation

Here we give a brief introduction to the Abstract Protocol notation [9]. 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 "$\mathbf{rcv}\ \langle message \rangle\ \mathbf{from}\ q$", (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 $\langle statement \rangle$ of an action of process p is a sequence of $\langle skip \rangle$, $\langle send \rangle$, $\langle assignment \rangle$, $\langle selection \rangle$, or $\langle iteration \rangle$ statements of the following forms:

$\langle skip \rangle$: **skip**
$\langle send \rangle$: **send** $\langle message \rangle$ **to** q
$\langle assignment \rangle \langle variable\ in\ p \rangle\ :=\ \langle expression \rangle$
$\langle selection \rangle$: **if** $\langle boolean\ expression \rangle\ \rightarrow\ \langle statement \rangle$
  ...
  □ $\langle boolean\ expression \rangle\ \rightarrow\ \langle statement \rangle$
  fi
$\langle iteration \rangle$: **do** $\langle boolean\ expression \rangle\ \rightarrow\ \langle statement \rangle$
  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: ESPs and banks. The major role of ESPs is to send and receive email for their users. We do not assume that every ESP has to join the Zmail protocol, although we expect that eventually most ESPs will. We call the ESPs that are running the Zmail protocol "*compliant ESPs*". 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 ESP 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 ESP and let each compliant ESP manage e-pennies for its own users.

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

process ESP[i:0..n-1]

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

inp B_b: **integer**, {public key of bank}
$limit$: **array** [0..m-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 ESP process has three constants n, m, and $compliant$, and all ESP processes share the same value for each of the three constants. Constant n is the number of ESPs. The n ESP processes are $ESP[0], ESP[1], \cdots, ESP[n-1]$. For simplicity, we assume each ESP 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 ESP is compliant. This array is maintained and published by the bank. When an ESP $ESP[j]$ 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 ESP. 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 ESPs}
$compliant$: **array** [0..n-1] **of boolean**
{array compliant indicates which ESP 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 ESP}
$verify$: **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 ESP $ESP[i]$ and the bank, during which we will explain the meaning of every input and variable of ESPs and the bank.

4.1 Zero-sum email transfer

In process $ESP[i]$, each user specifies the maximum number of emails that they can send out to compliant ESPs 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 $ESP[i]$ uses the array $sent$ to keep track of the number of emails that each user sends to compliant ESPs in each day. At the end of every day, array $sent$ is reset to 0. Process $ESP[i]$ maintains the balance of e-pennies for every user using the array $balance$. Process $ESP[i]$ records the transaction of e-penny exchanges with other ESPs in the array $credit$. The initial value of every element in array $credit$ is zero.

Process $ESP[i]$ maintains a variable called "$cansend$". When $cansend$ is true, process $ESP[i]$ can send out email. When process $ESP[i]$ sends an email to a compliant ESP $ESP[j]$, $credit[j]$ is increased by one; when process $ESP[i]$ receives an email from a compliant ESP $ESP[j]$, $credit[j]$ is reduced by one. Later we will show that the bank can detect misbehaved ESPs using the information in the $credit$ array of every ESP. The initial value of $cansend$ is true. The pseudocode of the process $ESP[i]$ 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.

□ $cansend \rightarrow$
$s :=$ **any**; $j :=$ **any**; $r :=$ **any**;
{user s of $ESP[i]$ wants to send email to user r of $ESP[j]$}

```
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 ESP[j]
             □ balance[s] = 0 ∨ sent[s] ≥ limit[s] → skip
             fi
           □ ∼ compliant[j] → send email(s, r) to ESP[j]
           fi
fi
```

The pseudocode for receiving email is as follows:

```
□ rcv email(s, r) from ESP[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 $ESP[i]$ 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 ESP, and process $ESP[i]$ maintains the balance of real pennies for every user using array $account$. A user can buy and sell e-pennies with their ESP. The pseudocode of the process $ESP[i]$ 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      := avail − 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      := avail + x;
    □ balance[t] < x →skip
    fi
```

4.3 Transaction With Bank

For the variable $avail$, process $ESP[i]$ specifies one lower bound named $minavail$ and one upper bound named $maxavail$. When $avail < minavail$, process $ESP[i]$ needs to buy some e-pennies from the bank; when $avail > maxavail$, process $ESP[i]$ needs to sell some e-pennies back to the bank. In the communication between the bank and the process $ESP[i]$ 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 $ESP[i]$ 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 ESP has an account of real money with the bank, and the bank stores the balance of real pennies of compliant ESPs in its array $account$. The pseudocode of the process $bank$ for managing transactions with compliant ESPs is as follows:

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

□ rcv sell(x) from ESP[g] →
    nr, y := DCR(R_b, x); {ESP[g] wants to sell y e-pennies}
    account[g] := account[g] + y;
    send sellreply(NCR(R_b, nr)) to ESP[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 $ESP[i]$ to $ESP[j]$ and all the email from $ESP[j]$ to $ESP[i]$ are received by these two ESPs, the value of $credit[j]$ in process $ESP[i]$ plus the value of $credit[i]$ in process $ESP[j]$ should be zero. Otherwise, at

least one of the two ESPs has misbehaved. The bank needs to gather the $credit$ array from every ESP periodically, say one time a month, and then detect the suspected misbehaved ESPs, based on which the bank may make further investigation. Because each compliant ESP 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 ESPs, we use a simple timeout method. When the bank wants to gather $credit$ arrays, it sends out a request message to every compliant ESP. When a compliant ESP 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 ESP sends its $credit$ array to the bank. Thereafter, the ESP 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 $ESP[i]$ 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 ESPs, not email users. An email user still can instruct their ESP 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 ESPs.

Every time the bank wants to verify the compliance of ESPs, it first sends request to every compliant ESP. After the bank receives the $credit$ array from every compliant ESP, the bank starts to verify that for every two compliant ESPs $ESP[i]$ and $ESP[j]$, the value of $credit[j]$ in process $ESP[i]$ plus the value of $credit[i]$ in process $ESP[j]$ 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 ESP[i];
                  □ ∼compliant[i] → skip
                  fi;
                  i := i + 1
      od;
      canrequest := false

□     rcv reply(x) from ESP[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 (http://www.lsoft.com/) and Majordomo (http://www.greatcircle.com/majordomo/). 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 ESP 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 acknowl-

edge 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, ESPs 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 disinfecting "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 ESPs. We have seen that in the Zmail protocol a non-compliant ESP can still send email to a compliant ESP, but a user in a compliant ESP may decide to segregate or discard email from non-compliant ESPs, or require any email from a non-compliant ESP to pass a spam filter. As more and more ESPs become compliant, more people would choose not to accept any email from a non-compliant ESP, which in turn causes more people to use compliant ESPs and more ESPs 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 (http://www.iana.org/) 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 also 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] I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, and C. D. Spyropoulos. An experimental comparison of naive bayesian and keyword-based anti-spam filtering with personal e-mail messages. In *Proceedings of SIGIR-2000*, pages 160–167, 2000.

[4] X. Carreras and L. Marquez. Boosting trees for anti-spam email filtering. In *Proceedings of the 4th International Conference on Recent Advances in Natural Language Processing*, 2001.

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

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

[7] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *Proc. of Crypto-92, LNCS 740*, pages 139–147, 1992.

[8] K. R. Gee. Using latent semantic indexing to filter spam. In *Proceedings of the 2003 ACM symposium on Applied computing*, pages 460–464, 2003.

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

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

[11] A. Kolcz, A. Chowdhury, and J. Alspector. The impact of feature selection on signature-driven spam detection. In *First Conference on Email and Anti-Spam (CEAS)*, July 2004.

[12] K. Li, C. Pu, and M. Ahamad. Resisting spam delivery by tcp damping. In *First Conference on Email and Anti-Spam (CEAS)*, July 2004.

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

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

[15] 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.

[16] K. Yoshida, F. Adachi, T. Washio, H. Motoda, T. Homma, A. Nakashima, H. Fujikawa, and K. Yamazaki. Density-based spam detector. In *Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 486–493, 2004.