DiffMail: A Differentiated Message Delivery Architecture to Control Spam

Zhenhai Duan
Computer Science Department
Florida State University
Tallahassee, FL 32306
Email: {duan}@cs.fsu.edu

Yingfei Dong
Dept. of Electrical Engineering
University of Hawii
Honolulu, HI 96822
Email: {yingfei}@hawaii.edu

Kartik Gopalan
Computer Science Department
Florida State University
Tallahassee, FL 32306
Email: {kartik}@cs.fsu.edu

Abstract

Unsolicited bulk electronic mail (spam) is increasingly plaguing the Internet Email users and deteriorating the value of Email as a convenient communication tool. Although many different spam control schemes have been proposed and deployed on the Internet, the proportion of Email spam seen on the Internet has been increasing in recent years. In this paper we argue that the difficulties in controlling spam can be attributed to the lack of user control on how different Email messages should be delievered on the Internet. In the current Email delivery architecture, a party can at will force another party to be involved in an Email communication, regardless of whether the latter is willing to accept the message. Based on this observation, we propose a differentiated message delivery architecture—DiffMail. In DiffMail, a user can classify Email senders into multiple classes and handle messages from each class differently. For example, the user may directly accept messages from regular contacts, while asking other senders to hold their messages at their own mail servers. Such messages can be retrieved from the sender's mail server *if* and *when* the receiver wishes to do so. In this way, DiffMail achieves several appealing objectives. Amongst others, regular correspondence is handled in the same way as in the current Email architecture, spammers are discouraged from blindly sending spam to arbitrary users, and it helps to improve the effectiveness of real-time blacklists of spammers. In this paper we present a detailed design of the DiffMail architecture and conduct empirical studies to illustrate its performance trade-offs using real-world Email archives. Furthermore, we also illustrate how DiffMail can be incrementally deployed on the Internet.

I. Introduction

Electronic mail (Email) is one of the most popular applications on the Internet. As the ability to send and receive Emails is being integrated into hand-held devices such as cell phones, we have increasingly come to rely on Email as an indispensable communication tool. However, the current Email delivery infrastructure also provides an essentially effort-free platform for spammers to send a deluge of unsolicited commercial messages, commonly known by the term *spam*. It is estimated that nowadays spam messages constitute 79% of all business Emails, up from 68% since the US federal Can-Spam Act of 2003 took effect in January 2004 [6]. In monetary terms, dealing with spam costed businesses more than $20 billion in 2003 [13]. Email spam is increasingly plaguing the Internet Email users and threatening to deteriorate the value of Email as a convenient communication tool. For example, a recent study showed that 29% of Email users have cut down their usage of Email [25].

Given the importance of controlling spam to preserve Email as a valuable communication tool, this issue has attracted a great amount of attention in both networking research and industry communities. These research and development efforts can be roughly classified into three categories—Spam filters [5], [9], [16], [17], [26], [28], [29], sender authentication schemes [11], [21], [23], and mechanisms to increase the cost of sending Email messages (sender-discouragement mechanisms) [18], [19]. Email spam filters try to identify spam based on the content of a message (content-based filters) or the IP addresses or domains of the sending machines (IP-address-based filters), so that users may not need to spend time on processing these messages. However, spam filters are essentially a *reactive* method, they can only at best adapt themselves according to the strategies used by spammers. Moreover, no content-based filters are perfect. Although these filters generally have a very high filtering rate (often higher than 99%), we still see many spam messages sneaking into our mailboxes. IP-address-based approaches block messages coming from known spammer sites by maintaining/consulting real-time blacklists (RBL). However, as spammers can frequently change the IP addresses of their mail servers and/or their Internet Service Providers (ISPs), how to quickly detect spammer sites and timely update RBL are increasingly becoming a major issue for any IP-address-based approaches.

As a promising way to prevent spammers from spoofing sender addresses, different sender authentication methods have been recently proposed, including Sender Policy Framework (SPF), Sender-ID, and DomainKeys [11], [21], [23]. Relying on special records published in DNS databases, a receiver mail server supporting this feature can verify if the sending machine is a legitimate mail server to send messages on behalf of the claimed sender. In this way, sender address spoofing is prevented to a certain degree. However, this method alone cannot stop spam, as spammers can publish their DNS records. Indeed, it was reported that 16% of messages in a recent collection of 400,000 spam were from spammers who have published their Sender-ID in the DNS databases to comply with Sender-ID like approaches [7].

Another method to control spam is to discourage spammers from sending a large number of messages, such as paid Email and challenge-response based mechanisms [18], [19]. Although the idea of paid Email is very simple, in practice there are many issues that may not be easy to address, for example, how much postage a user needs to pay for each Email, how to collect and distribute postage on the global Internet, etc. Moreover, ISPs may become large spammers [19]. On the other hand, challenge-response based schemes may take users too much time to send an Email, and consequently, may make Email a less attractive communication tool. We discuss other anti-spam mechanisms in Section VI.

In this paper we propose a novel differentiated message delivery architecture, DiffMail, which can effectively control spam messages while allowing normal messages to be handled in the same way as in the current Email delivery architecture. First, we note that the difficulties in controlling spam can be attributed to the lack of user control on how Email messages from different senders should be delivered on the Internet. In the current Email delivery architecture, a party can at will force another party to be involved in an Email communication, regardless of whether the latter is willing to accept the messages. In the early days of the Internet development, this was not a big problem as people on the network largely trust each other. However, since the commercialization of the Internet in mid-1990, the nature of the Internet community has changed. It becomes less trustworthy, and Email spam is possibly one of the most notable examples of the untrustworthy nature of the Internet [3].

In designing DiffMail, we want to achieve several important goals [14]:

- Receivers should have more control over how different messages are delivered from senders to the receivers, in order to control spam;
- Messages from regular correspondents should be handled in the same way as in the current Email delivery architecture, in order to preserve Email as a convenient communication tool;
- People other than regular contacts may be allowed to express the intention to communicate, in order to retain Email as an open communication tool;
- DiffMail must allow incremental deployment.

To achieve these goals in DiffMail, each receiving party[1] can classify senders into different classes, and specify how messages from different classes should be delivered to the party. Here a sender can be defined at different granularities, for example, Email accounts, IP addresses, and domain names (see Section III). Messages from different classes may be handled differently. For example, a user may directly accept messages from regular contacts, while asking other senders to hold their messages at their own mail servers. Such messages can be retrieved from the sender's mail server *if* and *when* the receiver wishes to do so. It is critical to note that a sender needs to maintain such messages on *his own mail server* before they are retrieved by the receivers.

DiffMail has several salient advantages in controlling spam while preserving Email as an open and convenient communication tool. First, and most importantly, by asking senders (non-regular contacts) to maintain messages on their mail servers, spammers are forced to keep their servers up. They cannot simply send a large number of spam messages, shut down their servers, and switch to another domain (and/or change IP addresses). In this way, DiffMail helps to improve the effectiveness of IP-address-based filtering schemes. Second, since a (complete) message is

[1]Throughout the paper, a party or a user (sender/receiver) can be either a Mail Transfer Agent [20] or a real Email user. We will distinguish them when it is desirable.

only retrieved by a receiver at his will, less bandwidth and storage resource will be consumed at the receiver side *if the user does not retrieve the majority of messages from non-regular contacts*. Third, spammers now have more responsibilities to maintain their outgoing messages, for example, deleting messages that have not be retrieved by receivers after a certain amount of time. This will discourage spammers from blindly sending spam to arbitrary user accounts. We will illustrate these advantages in the later sections as we present the architecture of DiffMail.

The remainder of the paper is structured as follows. In Section II we present an overview of the DiffMail architecture. Section III details the architecture, including its components and protocols. We discuss the incremental deployment issues in Section IV. Section V conducts empirical studies to illustrate the performance trade-offs in DiffMail using real-world Email archives. After presenting other related work in Section VI, we conclude the paper and discuss our ongoing work in Section VII.

II. DiffMail: Solution Overview

In this section we present a functional overview of the DiffMail architecture to highlight how it helps to control Email spam while still preserving Email as an open and convenient communication tool. We will discuss the details of the DiffMail architecture, including both its components and protocols, in the next section.

A. Sender Differentiation and Email Delivery Control

As discussed in the previous section, a party in the current Email architecture can at will send messages to another party without considering if the latter is willing to accept the messages. Users cannot differentiate senders and cannot control how messages from different senders should be delivered on the Internet, which make the system extremely vulnerable to spam.

To effectively control spam, users should be able to specify how they want to communicate with others. One such example is Instant Messenger (IM), where users are equipped with the capability to control who they want to communicate with. Although IM's closed communication model may help control spam, it is not adequate for Internet Email. The value of Internet Email greatly depends on its open communication model, which we need to preserve in designing any new Email architecture. Moreover, it is also important that regular correspondence should not be affected. If it takes much more efforts for regular users to send Email, or regular Email cannot be delivered in a timely fashion, the value of Internet Email as a convenient communication tool will be largely deteriorated.

In order to preserve Email as an *open and convenient* communication tool while being able to control spam, DiffMail allows users to differentiate Email senders and control how messages from different senders should be delivered. Before we present the basic idea of DiffMail, we first define a notation–*Sender Email Address Domain (SEAD)*. SEAD is used to represent sender(s) of Email messages. It can be defined at different granularities. In this paper we consider three granularities: Email accounts (in the form of *useraccount@domainname*), IP addresses, and network domain names. An Email account uniquely identifies a single sender; an IP address designates all the senders whose Email messages are sent from this IP address, and a network domain name represents all the senders within this domain;

In DiffMail, a user is able to classify SEADs into several groups or classes, which are defined and managed by the users. Associated with each class is an action, which indicates how messages from the class should be handled regarding their deliveries. Below we present an example SEAD classification. This system has three classes, given in the form of **class name/action**:

- *well-known spammers/block*: messages from this class (spammers) are *not accepted*. They are never delivered from the sender to the receiver.
- *regular contacts/accept*: messages from this class will be handled in the same way as in the current Email architecture. In particular, complete messages (including both headers and bodies) are delivered directly from the sender to the receiver.
- *unclassified sources/partial*: this class includes all senders that belong to neither *regular contacts* nor *well-known spammers*.

Unlike messages from regular contacts or well-known spammers, the ones from *unclassified sources* can be either spam or regular messages. Hence this represents the most critical category to manage in effectively controlling spam. To discourage spammers we should prevent their messages from being delivered to the receivers. On the

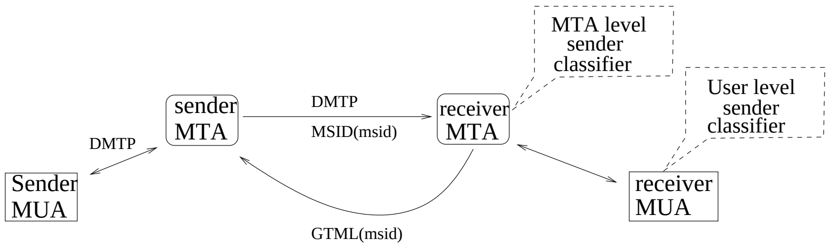

Fig. 1. Illustration of DiffMail architecture.

other hand, we should also provide legitimate users (who are not in the regular contact class yet) with a way to express the intention to communicate. To balance these two considerations, DiffMail only accepts the envelopes of messages from this class [20], [27]. The complete messages are *required to be stored at the sender Mail Transfer Agents (MTAs)*. If a user wants to read such messages, he can retrieve the messages from the sender MTAs at a later convenient time. The implication of handling messages from unclassified sources is two-fold. First, by only delivering the envelope of a message from sender to receiver, less bandwidth, storage, and time will be occupied at the receiver side, which may be especially important for dial-up users. On the other hand, if the user indeed wants to read the message from an unclassified source, some extra bandwidth and time will be used. However, users will most unlikely be interested in messages from unclassified sources, and therefore, the majority of such messages will not be retrieved.

By differentiating messages coming from different classes, DiffMail accomplishes several appealing goals. First, messages from regular contacts will be handled in the same way as in the current Email architecture, no extra efforts are needed for sending such messages. This will preserve Email as a convenient communication tool. Secondly, although messages from unclassified sources cannot be directly delivered, DiffMail provides a means for such senders to express the intention to communicate. In this way, we can retain Email as an open and generic communication tool. And thirdly, receivers have more control over how they want to communicate with the rest of the world.

B. Outgoing Email Management

In DiffMail, sender MTAs have to store messages before they are retrieved by the receivers, or explicitly deleted by the senders. Therefore senders in DiffMail have more responsibility to manage their outgoing Email messages compared with that in the current Email architecture. This is especially true for spammers, who are most likely in the well-known spammer classes or the unclassified classes of others. For both these two cases, the senders need to manage their undelivered messages stored on their own MTA servers. This has several advantages. First, spammers need more efforts to send spam because they need to delete undelivered messages on the mail server; Second, and more importantly, spammers cannot simply shutdown their mail servers after sending out a large number of messages. They need to wait for receivers to retrieve the messages. This helps to identify spammers and improve the effectiveness of IP-address based filtering schemes, which rely on IP addresses to block spam messages. If spammers can frequently change their IP addresses or domain names, the effectiveness of such schemes is limited.

III. DiffMail: Architecture, Components, and Protocol

In this section we present in detail the DiffMail architecture. We discuss the components of DiffMail and the protocols used among them to manage messages and to deliver them from senders to receivers. Incremental deployment issues will be discussed in the next section. Figure 1 illustrates the basic architecture of DiffMail.

A. Message Composition and Local Management

A sender uses some Mail User Agent (MUA) to compose the messages that he wants to send [20]. After a message is composed by the sender, the sender delivers the message to the sender side Mail Transfer Agent (MTA) using an extended version of the Simple Mail Transfer Protocol (SMTP) [20]. We refer to this extended version of SMTP as Differentiated Mail Transfer Protocol, or simply DMTP. DMTP extends SMTP in two aspects: it allows

Well-known spammers/block
- $a_1.b_1.c_1.d_1$
- $a_2.b_2.c_2.d_2$
- spammer1.com
- spammer2.com

Good citizens/accept
- $x_1.y_1.z_1.w_1$
- $x_2.y_2.z_2.w_2$
- $x_3.y_3.z_3.w_3$
- citizen1.com
- citizen2.com

Unclassified sources/partial
- *

Fig. 2. Illustration of MTA classifier.

Regular contacts/accept
- user1@comcitizen1.com
- user2@comcitizen1.com
- user3@educitizen1.edu
- user4@educitizen2.edu
- user5@govcitizen.gov

Fig. 3. Illustration of user classifier.

senders to manage their outgoing message folders (see below), and it supports message retrieval by receivers (see Subsection III-B). For simplicity, we refer to a sender side MTA server as an SMTA, and a receiver MTA server as an RMTA.

All the outgoing messages are stored at the SMTA. For this purpose, the SMTA maintains an outgoing message folder for each sender. This folder contains all the user's messages that have not been delivered and have not been deleted. A user can explicitly delete his outgoing messages from the SMTA folder by means of DMTP. An SMTA can also delete messages on behalf of the users. However, an SMTA can only delete an outgoing message after it has been delivered to *all the intended receivers or after a certain user-configurable expiry time*.

B. Message Delivery between MTAs

An SMTA communicates with an RMTA using DMTP, trying to deliver messages from the sender to the receiver. As discussed in the previous section, receivers in the DiffMail framework have more control on message deliveries than in the current Email architecture. A receiver differentiates the treatment of messages from different senders by defining a number of sender classes or groups. We refer to this functionality module as sender classifier. Sender classifiers can be defined at two levels: MTA level and end user level (Figure 1). For clarity, we first describe a base model where only the MTA level sender classifier is defined and used to control message deliveries between MTAs. We will later extend the base model to include both MTA level and end user level sender classifiers.

1) Base Model—MTA classifier and Message Delivery between MTAs: We illustrate the definition of MTA classifier using the same three-class example we have seen in Section II. Namely, senders are classified into three classes: well-known spammers, regular contacts (or good citizens), and unclassified sources. However, in an MTA classifier, senders are in general only defined at IP-address level or domain-name level. That is, an MTA only tries to classify senders based on the IP addresses or domain names of the sender MTAs. Figure 2 presents an example of the MTA level sender classifier, where $a_i.b_i.c_i.d_i$ and $x_i.y_i.z_i.w_i$ represent IP addresses (both the IP addresses and domain names in the figure are only for illustration purpose).

Now let us discuss how an RMTA handles message delivery requests and how MTA classifier is used to enforce message delivery controls (see Figure 4). The RMTA listens on port **25** for message delivery requests from sender MTAs, which express the requests by opening a TCP session on the port. When the RMTA receives a request, it will fetch the IP address and domain name of the SMTA (a DNS lookup may be issued). Then the RMTA consults its MTA classifier to determine how to treat the request. If the IP address (or domain name) belongs to a well-known spammer, the request is blocked immediately: the RMTA will reply with the code **550 (permanent error, don't retry)**, and then close and clean the TCP session. (All the commands and reply codes used in DMTP are defined in SMTP [20], except the ones listed in Table I, which are proposed in our DMTP.) If the IP address (or domain name) belongs to the good citizen class, the RMTA will proceed in the same way as if SMTP is used. In particular, the SMTA is allowed to issue the **DATA** command to deliver the complete message to the RMTA.

TABLE I

NEWS COMMANDS/REPLY DEFINED IN DMTP.

Commands/Replies	Explanation
MSID	For SMTA to inform RMTA about the reference to a message
GTML	For RMTA to retrieve a message from SMTA
253	For RMTA to inform SMTA to send reference (**MSID**) instead of message (**DATA**)

```
Require: WKS: well-known spammer class;
Require: GCC: good citizen class;
 1: ip = SMTA's IP address;
 2: dn = SMTA's domain name;
 3: if (ip $\in$ WKS) OR (dn $\in$ WKS) then
 4:    /* well-known spammers */
 5:    reply with 550;
 6:    close TCP session;
 7: else if (ip $\in$ GCC) OR (dn $\in$ GCC) then
 8:    /* good citizens */
 9:    reply with 220 (to TCP session opening request);
10:    proceed as if SMTP used;
11: else
12:    /* unclassified sources */
13:    reply with 253;
14:    accept MSID command;
15:    reject DATA command;
16: end if
```

Fig. 4. Base model message delivery algorithm.

```
Require: WKS: well-known spammer class of MTA;
 1: ip = SMTA's IP address;
 2: dn = SMTA's domain name;
 3: if (ip $\in$ WKS) OR (dn $\in$ WKS) then
 4:    reply with 550;
 5:    close TCP session;
 6: else
 7:    proceed to the MAIL command;
 8:    sender = Email address of sender;
 9:    proceed to the RCPT command;
10:    RCC = regular contact class of the receiver;
11:    if sender $\in$ RCC then
12:       reply with 250 (to RCPT command);
13:       proceed as if SMTP used;
14:    else
15:       /* unclassified sources */
16:       reply with 253;
17:       accept MSID command;
18:       reject DATA command;
19:    end if
20: end if
```

Fig. 5. Message delivery algorithm in extended model.

If the IP address (and domain name) does not belong to the good citizen and well-known spammer classes, the RMTA classifies it as an unclassified source. For this case, only partial information is delivered from the sender to the receiver. In particular, the SMTA is not allowed to issue the **DATA** command to deliver the message. Instead, the receiver will retrieve the message if he is interested in it. For this purpose, the RMTA will respond to the request with a new reply code **253**, which has the following meaning:

> **253**: the message will not be received immediately, don't send **DATA** command. Instead, use the **MSID** command (see below).

In this case, the SMTA will store the message locally and generate a reference or index to the message. The index to the message is sent to the RMTA using a new command **MSID**. The **MSID** command looks like this:

> **MSID**: msid; subject (optional)

Where *msid* is the reference or index to the stored message; and *subject* is a brief text saying what the message is about. Note that *subject* is optional. Moreover, DiffMail also requires that the MSID command line cannot exceed a configurable maximum length.

If the user wants to read the message, the *receiver MTA* will retrieve the message on behalf of the user. When the RMTA needs to retrieve a message, the *get mail* command **GTML** is issued, which has the following syntax:

> **GTML**: msid; receiver

Where *msid* is the index to the message to be retrieved, and *receiver* is the Email address of the receiver. For security reasons, when the SMTA receives the **GTML** command, it needs to verify that the corresponding message is for the receiver, and more importantly, the requesting MTA is the mail server responsible for the receiver (i.e.

the one which was originally contacted for message delivery). The latter can be verified through a DNS lookup as in the current Email architecture. We discuss potential security concerns in Subsection III-D.

Figure 4 summarizes the message delivery procedure between MTAs using the base model.

2) Extended Model—User Level Sender Classifier: Now let us extend the base model to include both MTA level and user-level sender classifiers (or simply user classifiers). User classifiers differ from MTA classifiers in two aspects. First, unlike MTA classifiers where senders are defined at the IP address level or domain name level, senders in user classifiers are specified using Email addresses in the form of *username@domainname*; and secondly, we do not need to enumerate all classes of senders in user classifiers. For example, considering again the three-class sender example, which include well-known spammers, regular contacts, and unclassified sources. In a user classifier, we may only need to specify the regular contact class. Note that users may not have sufficient knowledge to specify a spammer, and more importantly, the user accounts of a spammer's Email addresses are normally randomly generated. So, it may not be effective to let end users to specify spammers at Email address level. Figure 3 illustrates an example of user classifiers, where only regular contacts are defined. Note that since user classifiers are defined at Email address level, it is important that Email address spoofing should be prevented. Therefore, some schemes such as Sender ID, DomainKeys, or SPF should be in place. Note that these schemes cannot completely prevent Email address spoofing, but they provide us with some confidence in authenticating the addresses.

Before we proceed to describe how a user classifier could be used along with MTA classifiers to exert message delivery control, it is worth noting that *user classifiers are optional, an institution may not support user classifiers, or only allow certain users to have this feature*. Below we assume both user classifiers and MTA classifiers are used.

When we have both MTA classifiers and user classifiers defined, a critical problem we need to resolve is the priority of the classes defined in these two classifiers. The priority of the classes determines the order of the classes and consequently the actions taken to handle the message deliveries. DiffMail allows individual institutions to define their own priorities relying on their own policies. To illustrate how the two classifiers can be used together, below we define a simple priority: MTA classifier's well-known spammer class has a higher priority than user classifier's regular contact class. All senders not in these two classes are considered as *unclassified sources*. Using this priority, let us now see how message delivery requests are handled when both MTA classifiers and user classifiers are supported (Figure 5). When a message delivery request is received, the RMTA will first check if the IP address (and domain name) of the SMTA is in the well-known spammer class. If so, the mail server rejects the request by responding with the reply code **550**. Otherwise, the mail server starts the conversation with the SMTA. When it gets the sender's Email address from the command **MAIL**, it checks if the address is in the regular contact class in the user classifier. If it is the case, the transaction will proceed in the same way as if SMTP is in place. In particular, the SMTA can directly issue the **DATA** command to send the complete message to the RMTA. Otherwise, the sender belongs to the unclassified source class. In this case, the RMTA will respond to the MAIL command with the reply code **253**, informing the SMTA that the message cannot be delivered directly. Instead, the receiver will retrieve the message later. The transaction will proceed with the SMTA issuing the **MSID** command instead of the **DATA** command. Figure 5 summarizes the algorithm of controlling message deliveries using both MTA and user classifiers.

C. Retrieving Messages by Receivers

As we discussed above, a complete message (including header and body) is directly delivered from the SMTA to the RMTA only if the message is from a *regular contact* (good citizen). Messages from a well-known spammer are blocked immediately. Such messages are never delivered from the SMTA to the RMTA. Now let us examine how a message from an unclassified source is handled at the receiver.

Remember that complete messages from unclassified sources are not delivered from the SMTA to the RMTA directly. Instead, only the envelope of such a message is delivered. For ease of exposition, we refer to the messages only containing envelopes as *short messages*. For a short message, the RMTA will try to identify if this is a spam message by parsing the subject (if present) contained in the **MSID** and detecting anomalies in the envelope. If a spam message is identified, it is marked so. When the receiver retrieves messages from his MTA server, all the incoming messages are delivered to his MUA, including short messages. (Actually the receiver MUA gets messages from POP, IMAP, or other post office servers [24], [10]. For the sake of simplifying the presentation, we ignore this detail.) For a short message, the RMTA provides the receiver with a mechanism to indicate if he wants to retrieve the complete message. If the receiver wants, the *receiver MTA* will contact the SMTA to retrieve the message using the **GTML** command, with the corresponding message ID, *msid*, coming with the **MSID** command from the SMTA. After the RMTA gets the complete message, it will try to determine if the message is spam using some content-based spam filters. If indeed the message is a spam, it is marked so before it is delivered to the receiver MUA. Furthermore, if many spam messages are received from an MTA, the corresponding IP address (and domain name) of the MTA may be added to the *well-known spammers* class.

If a receiver does not want to accept the complete message, the short message will be directly deleted by the receiver. The short message is also implicitly removed from the RMTA. Note that, the RMTA does not have the obligation to inform the SMTA that the receiver does not want to accept the message.

D. Discussions

1) Security Concerns about Message Retrieval: DiffMail does not allow messages from unclassified sources to be delivered from the SMTAs to the RMTA directly. Instead, the receivers will retrieve such messages (through their MTAs). We argue that the potential security issues arising from this model is not acute. Note first that important messages are normally communicated amongst regular contacts. Such messages will be handled in DiffMail in the same way as in the current Email architecture. Secondly, individual users cannot retrieve messages directly, they have to rely on their corresponding MTAs to retrieve the messages. These MTAs can be authenticated through the **MX** records in the DNS servers. Thirdly, the *msid*s are generated randomly based on the messages, they cannot be easily guessed. On the other hand, if the msid itself can be intercepted during transmission, so can be the entire message. Therefore, the proposed DiffMail model provides a security level comparable to that in the current Email architecture.

2) Mailing Lists: We believe that in the future all mailing lists will be mediated and some content-based spam filters will be deployed by each mailing list. It will become harder to send spam messages through mailing lists. In DiffMail, we suggest all the users to add into the *regular contacts* class the addresses of the mailing lists that he has joined. In this way, messages from other mailing lists can be handled with extra care.

3) Populating Regular Contact Class: It is clear that a user can add into the regular contact class his current regular correspondents, for example, the ones in his address book. However, it is conceivable that a receiver may want to communicate with someone who is currently not in the regular contact list, for example, an old classmate who has not contacted the receiver for years. We consider two ways to handle this. One is through the normal DiffMail message handling procedure. In this case, the address will be classified into the unclassified source class. And eventually, the receiver may read the complete message and finds out who the sender is. However, given that a receiver may pay less attention to messages from unclassified sources, it may take a long time for this person to be added to the regular contact class. The second is some out-of-band approaches, for example the ones used by some of the current email servers to register a new user and/or send messages. In this approach, the sender needs to send the message through some web-based interface. Moreover, some mechanism is used to ensure that automatic email agent cannot fill the web forms and send messages (e.g., by asking users to solve some puzzles).

This web-based email agent can be either managed by some well-known web sites, or by the local network of the receiver. In either case, the RMTA will directly accept the complete messages and mark them as *OUTOFBAND* when they are forwarded to the user MUA. In this way, the users may handle them promptly.

4) Exporting User Regular Contacts to Service Providers: Note that users may not be willing to export their own regular contact list to the service providers. This may especially be the case for home users who are served by some ISPs. Should user-level sender classifiers be supported, some mechanisms such as bloom filters (which may incur some false positive) can be employed to conceal the exact identifications of the regular contacts of the users [4].

IV. Incremental Deployment

In this section we outline an approach that DiffMail can be incrementally deployed on the Internet. We assume that the MTA in consideration supports DiffMail, i.e., adopts the DMTP protocol, and show how it interacts with

Algorithm 1 DiffMail RMTA's algorithm for incremental deployment.

```
Require: WKS: well-known spammer class;
Require: GCC: good citizen class;
 1: ip = sender MTA's IP address;
 2: dn = sender MTA's domain name;
 3: if (ip ∈ WKS) OR (dn ∈ WKS) then
 4:    reply with 550;
 5:    close TCP session;
 6: else if (ip ∈ GCC) OR (dn ∈ GCC) then
 7:    reply with 220 (to TCP session opening request);
 8:    proceed as if SMTP used;
 9: else
10:    /* unclassified sources */
11:    reply with 220 (to TCP session opening request);
12:    proceed to the EHLO command;
13:    if (found keyword "DiffMail") then
14:       /* sender supports DiffMail */
15:       proceed according to DiffMail's DMTP;
16:    else
17:       /* sender does not support DiffMail */
18:       respond to DATA command with 354;
19:       receive message;
20:       respond with 550 (permanent error);
21:       store message, send puzzle;
22:       message invisible to user;
23:       /* message becomes visible to user only after puzzle solved*/
24:    end if
25: end if
```

the rest of the world. Note that DMTP protocol can inter-operate with the current SMTP protocol. For simplicity, we only consider the the base model. User level sender classifiers are not used.

A. Informing Protocol in Use

In order to support incremental deployment, RMTA supporting DiffMail needs to know if the SMTA also supports DiffMail. If so, they can proceed according to DiffMail's DMTP protocol. Otherwise, RMTA needs to handle the message differently using the algorithm in the following subsection. For this purpose, an SMTA supporting DiffMail will inform the RMTA this fact by including keyword "DiffMail" in the greeting command **EHLO** (or **HELO**), as illustrated in the following example.

```
SMTA makes a request (connecting to TCP port 25)
RMTA: 220 ourdomainname.com
SMTA: EHLO citizen1.com DiffMail
```

In this example, we have assumed SMTA is not a well-known spammer. Otherwise, it will be blocked immediately, regardless of whether it supports DiffMail. Let us consider two cases. If the RMTA does not support DiffMail, the keyword "DiffMail" will have no effect, and the message can be delivered in the same way as in the current Email architecture (note that DMTP is a superset of SMTP). Now we consider the second case: the RMTA supports DiffMail. In this case, it will search for "DiffMail" in the **EHLO** command. If it finds the keyword, it knows that the SMTA supports DiffMail, and the transaction will proceed accordingly. Otherwise, the RMTA knows that the SMTA only supports SMTP, it does not understand DiffMail.

B. *Delivering Messages*

Now let us see how an RMTA (supporting DiffMail) handles a message delivery request from an SMTA. If the SMTA is a good citizen (in the *good citizen* class), the RMTA will accept the request (and the message) no matter the SMTA supports DiffMail or not. We now consider the case where the SMTA is an unclassified source. If the SMTA supports DiffMail, the RMTA will respond with reply code **253** to inform the SMTA to issue the **MSID** command instead of the **DATA** command. They will proceed according to the DMTP protocol.

If the SMTA does not support DiffMail, we need to process the request in a different way. Our basic principle is the same: We will not accept messages from unclassified sources directly. For such sources, some extra homework needs to be done by the sender before the messages can be received. This mechanism is meant to prevent spammers from using legacy SMTP as a backdoor to sneak in their spam emails. We adopt the challenge-response-like method [18]: the RMTA will respond to the **DATA** command with reply code **354 (start mail input)**. After the RMTA receives the *message body* from an unclassified source (which does not support DiffMail), the RMTA stores the message locally in a temporary location and responds with the reply code **550** to inform the SMTA about a permanent error so that the SMTA will not retry. However, the RMTA will automatically send a message to the sender requiring him to solve a puzzle. The message will be delivered to the receiver from the RMTA only if the sender can solve the puzzle. DiffMail does not dictate a specific form of puzzles. Any puzzles satisfying the following requirement can be used by the RMTA: *It cannot be solved by machine automatically*. We need this requirement to prevent spammers from relying on machines to automatically solve puzzles and generating replies. Algorithm 1 summarizes the algorithm for an RMTA to handle requests from the two types of SMTAs. Note that we assume the RMTA supports DiffMail.

Spammers normally do not have the capability to solve puzzles for each message they send. So this approach will help us control Email spam. On the other hand, if the sender is legitimate but not in the regular contact class yet, this method puts some extra burden for him to send a message. However, such senders will be added to the regular contact class eventually and thereafter they do not need to take extra efforts to communicate. This approach introduces some extra overhead at RMTAs as they need to send out puzzles and check the answers. To prevent RMTAs from being overloaded with such tasks, we can prioritize the tasks of the RMTAs [30]. For example, RMTAs only handle puzzle-related tasks when there is no new message request.

Before we leave this section, we note that there are two incentives for a site to deploy DiffMail. First, it may want to deploy DiffMail to control incoming spam. Secondly, their local users may not want to bear the burden to solve puzzles. Rather, they would like the mail server to support DiffMail (**MSID**) to store the message on the server. Over time, we believe that increasing use of DMTP would force legacy SMTP users to switch to DMTP.

V. Empirical Studies

In this section we conduct a number of empirical studies to illustrate the costs and performance gains of the DiffMail architecture. As a first step towards understanding the performance trade-offs in DiffMail, we focus on two performance metrics: Distribution of the number of regular contacts of a user and distribution of spam message *body* lengths. Given that users of DiffMail may need to maintain regular contact classes, the number of regular contacts refects the extra burden that users may need to bear in using DiffMail (if user-level sender classifiers are in use). On the other hand, bodies of messages from *unclassified sources* will not be directly delivered from senders to receivers. It will result in bandwidth and storage savings if users do not retrieve such messages (or the majority of such messages). The distribution of spam message body lengths provides us with an indication on the savings a user gains. Before we present the empirical study results, let us first discuss the data sets using in the studies.

A. *Data Sets*

We utilize data from two sources. The first set of data is from the University of Leipzig, Germany [12]. This set of data contains the log of emails in and out of the university from 9/2/2001 to 11/28/2001[2]. The log records the following information of a message: date, time, anonymized sender and receiver addresses. Addresses are also

[2]It appears that there are no recorded messages in some days. This could be caused by recording artifact, or there was indeed no message in those several days. But this should not have significant impact on our observations even if it is because of recording artifact.

TABLE II

NUMBER OF MESSAGES IN EACH SPAM ARCHIVES

Messages	5/15	6/15	7/15	8/15	9/13	10/15
Complete	478	2742	415	346	462	393
Damaged	4	2	0	2	1	5
Total	482	2744	415	348	463	398

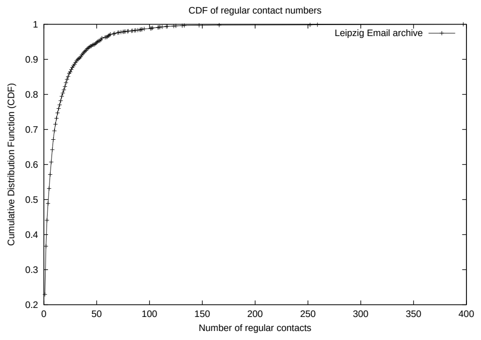

Fig. 6. CDF of regular contact numbers.

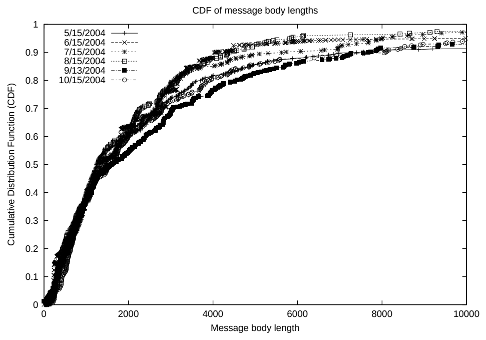

Fig. 7. CDF of message body lengths.

distinguished as being "internal" or "external" depending on if it is in the university domain. This data set contains a total of $447,543$ messages. We refer to this date set as "Leipzig Email Archive."

The second data set is from the Spam Archive site [1]. This site maintains archives of Email spam contributed by Internet Email users. We (randomly) select spam archived by the site on 5/15/2004, 6/15/2004, 7/15/2004, 8/15/2004, 9/13/2004 (no archives on 15th and 14th of the month), and 10/15/2004. Due to forwarding problems, some messages in the archives are damaged or incomplete, we remove such messages from the data sets before we analyze the data. Table II shows the total number of messages and the number of complete messages. We refer to the data set *after excluding the incomplete messages* as a "Spam Archive."

B. Distribution of the Number of Regular Contacts

In the Leipzig Email Archive data set, we are not able to distinguish messages from spammers and those from regular contacts. Because of this, we only consider the addresses to which a user sent messages and regard this set of addresses as regular contacts of the user. Moreover, we only compute regular contacts for users who belong to the university (i.e., internal users). Note that such users may send messages to both internal and external users. Note also that this may only represent a lower bound on the number of regular contacts for the users. We observe $1,753$ internal users who sent messages in the data set. Figure 6 presents the cumulative distribution function (CDF) of the number of regular contacts for such users. From the figure we can see that more than 90% of users have less than 35 regular contacts, which we believe holds for general Internet Email users. Therefore maintaining regular contact class should only require minor efforts from users given the relatively small numbers of regular contacts.

C. Distribution of Message Body Lengths

In this subsection we use the Spam Archive dataset to study the distribution of message body lengths. Note that we exclude message headers when we compute this length. Figure 7 depicts the CDFs of spam message body lengths. We can see from the figure that more than half of spam have a body longer than 1.5 KB, and 60% of spam longer than 2 KB. Although the bandwidth and storage saving from without retrieving a single message may not be significant, the overall potential savings will be promising considering the massive volume of spam on the Internet.

VI. RELATED WORK

Among all the proposed Email architectures that we are aware of, the Internet Mail 2000 (IM2000) project is probably most close to the DiffMail architecture [2]. In IM2000, all outgoing messages are stored at the sender

ISPs, and receivers retrieve messages from sender ISPs. To a certain degree, it is similar to the way how DiffMail handles messages from *unclassified sources*. However, IM2000 is mainly motivated by some management costs and difficulties in the current Email architecture such as storage cost at the receivers and difficulties to track message delivery status. On the other hand, DiffMail aims to control Email spam. More importantly, users in IM2000 needs to retrieve all the message sent to him from sender ISPs, regardless where the messages come from. This may raise more issues related to timely delivery, security, and incremental deployment. In contrast, DiffMail differentiates the treatment of messages from different senders. Messages from regular contacts will be directly delivered from the sender to the receiver in the same way as in the current Email architecture. Only messages from unclassified sources will be stored at the sender MTAs and retrieved later by the receivers. Furthermore, as stated in the project website, many design and implementation issues of IM2000 have not been addressed such as receiver notification and message delivery.

The Internet Message Access Protocol (IMAP) allows a user to retrieve part of a message, such as the message header without fetching the complete message, from *his mail server* [10]. However, it works only between the user's MUA and his local mail server. The complete message is first delivered from the sender MTA to the receiver MTA.

Email Prioritization was proposed in [30] as a way to control the impact of spam on legitimate messages. However, the performance of the system depends on how well it can predict that an incoming message is spam. Moreover, spammers still have the incentive to send a large number of messages given that the entire messages including both headers and bodies are still delivered from the sender to the receiver (even though they may do so at the cost of purchasing more machines). Li, Pu, and Ahamad proposed a method to slow down spam delivery by damping the corresponding TCP sessions [22]. However, it has the similar performance constraint to the Email prioritization method. Moreover, it is not clear what the long-term impact it will have to modify the behavior of TCP for a specific application, and if the spammers will respond by changing sender MTA's TCP behavior.

Gburzynski and Maitan proposed to use Email aliases to fight Email spam [15], where different Email aliases can be created for different purposes and used over a specific duration. However, its effectiveness relies on hiding Email addresses and their aliases. Moreover, users have more burdens to manage their accounts. For example, they need to create Email aliases and disseminate them to intended correspondents.

VII. Conclusion and Ongoing Work

In this paper we proposed and studied a differentiated messsage delivery architecture, DiffMail, to control Email spam on the Internet. By designating message delivery controls to receivers, DiffMail has several appealing features, among which the most important ones are that regular correspondence is handled in the same way as in the current Internet delivery architecture, Email senders have more responsibility to manage their outgoing messages. DiffMail discourages spammers from blindly sending spam to arbitrary users, as users will most unlikely retrieve the messages from spammers. Morever, it also helps to improve the effectiveness of real-time blacklists of spammers, as spammers now cannot frequently change the IP addresses and/or domain names of their mail servers. We conducted empirical studies to illustrate the cost and performance gains of DiffMail using real Email (spam) logs. Moreover, we also discussed how DiffMail can be incrementally deployed on the Internet.

Currently we are developing a prototype of DiffMail based on Sendmail and Procmail [8], [31]. We are also modeling the DiffMail system to theoretically study its performance.

References

[1] Apam Archive. Donate your spam to science. http://www.spamarchive.org/.
[2] D. Bernstein. Internet mail 2000 (IM2000). http://cr.yp.to/im2000.html.
[3] M. Blumenthal and D. Clark. Rethinking the design of the internet: the end-to-end arguments vs. the brave new world. *ACM Transactions on Internet Technology*, 1(1), August 2001.
[4] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. In *Proceedings of the 40th Annual Allerton Conference on Communication, Control, and Computing*, 2002.
[5] X. Carreras and L. Márquez. Boosting trees for anti-spam email filtering. In *Proceedings of RANLP-01, 4th International Conference on Recent Advances in Natural Language Processing*, Tzigov Chark, BG, 2001.
[6] T. Claburn. Big guns aim at spam. *Information Week*, March 2004.
[7] T. Claburn. Spammers hijack sender id. *Information Week*, September 2004.

[8] Sendmail Consortium. Welcome to sendmail.org. http://www.sendmail.org/.
[9] L. Cranor and B. Lamacchia. Spam! *Communications of the ACM*, 41:74–83, August 1998.
[10] M. Crispin. Internet message access protocol - version 4rev1. *IETF RFC3501*, March 2003.
[11] M. Delany. Domain-based email authentication using public-keys avertised in the DNS (domainkeys). Internet Draft, August 2004. Work in Progress.
[12] H. Ebel. The data of the e-mail network. http://www.theo-physik.uni-kiel.de/ ebel/email-net/email%5Fnet.html.
[13] The Editors. Product of the year: Spam? *Information Week*, January 2004.
[14] B. Gates. Preserving and enhancing the benefits of email. Executive E-mail, Microsoft Corporation, June 2004.
[15] P. Gburzynski and J. Maitan. Fighting the spam wars: A remailer approach with restrictive aliasing. *ACM Transactions on Internet Technology*, 4(1):1–30, February 2004.
[16] P. Graham. Better bayesian filtering. *http://www.paulgraham.com/better.html*, January 2003.
[17] P. Graham. A plan for spam. *http://www.paulgraham.com/spam.html*, January 2003.
[18] A. Juels and J. Brainard. Client puzzles: A cryptographic defense against connection depletion attacks. In *Proceedings of NDSS-1999 (Networks and Distributed Security Systems)*, February 1999.
[19] RISHI V K. Free lunch ends: e-mail to go paid. *The Economic Times*, February 2004.
[20] J. Klensin. Simple mail transfer protocol. RFC 2821, April 2001.
[21] M. Lentczner and M. W. Wong. Sender policy framework (spf): A convention to describe hosts authorized to send SMTP traffic. Internet Draft, February 2003. Work in Progress.
[22] K. Li, C. Pu, and M. Ahamad. Resisting spam delivery by TCP damping. In *Proceedings of First Conference on Email and Anti-Spam (CEAS)*, July 2004.
[23] J. Lyon and M. Wong. Sender ID: Authenticating e-mail. Internet Draft, August 2004. Work in Progress.
[24] J. Myers and M. Rose. Post office protocol - version 3. RFC 1939, May 1996.
[25] TechWeb News. Report: Spam causing web users to abandon e-mail. *Information Week*, March 2004.
[26] RBL. Real-time spam black lists (rbl). http://www.email-policy.com/Spam-black-lists.htm.
[27] P. Resnick. Internet message format. RFC 2822, April 2001.
[28] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.
[29] SpamAssassin. The apache spamassassin project. http://spamassassin.apache.org/.
[30] R. Twining, M. Williamson, M. Mowbray, and M. Rahmouni. Email prioritization: reducing delays on legitimate mail caused junk mail. In *USENIX Conference*, June 27–July 2 2004.
[31] S. van den Berg. Welcome to procmail.org. http://www.procmail.org/.