Email Prioritization: reducing delays on legitimate mail caused by junk mail

Richard Daniel Twining, Matthew M. Williamson,
Miranda Mowbray, Maher Rahmouni
Information Infrastructure Laboratory
HP Laboratories Bristol
HPL-2004-5
January 12th, 2004*

E-mail: {dan.twining, matthew.williamson, miranda.mowbray, maher.rahmouni}@hp.com

In recent years the volume of junk email (spam, virus etc.) has increased dramatically. These unwanted messages clutter up users' mailboxes, consume server resources, and cause delays to the delivery of mail. This paper presents an approach that ensures that non-junk mail is delivered without excessive delay, at the expense of delaying junk mail. Using data from two Internet facing mail servers, we show how it is possible to predict whether the next message sent from a particular server will be good or junk, by monitoring the types of messages previously sent. The prediction can be used to delay acceptance of junk mail, and prioritize good mail through the mail server, ensuring that loading is reduced and delays are low, even if the server is overloaded. The paper includes a review of server-based anti-spam techniques, and an evaluation of these against the data. We develop a model of mail server performance, and use it to predict the performance of the prioritization scheme. We also describe an implementation on a standard mail server.
Email Prioritization: reducing delays on legitimate mail caused by junk mail

Dan Twining, Matthew M. Williamson, Miranda J. F. Mowbray, Maher Rahmouni
HP Labs, Filton Road, Stoke Gifford, Bristol, BS34 8QZ, UK
{dan.twining.matthew.williamson, miranda.mowbray, maher.rahmouni}@hp.com
January 8, 2004

Abstract

In recent years the volume of junk email (spam, virus etc.) has increased dramatically. These unwanted messages clutter up users’ mailboxes, consume server resources, and cause delays to the delivery of mail. This paper presents an approach that ensures that non-junk mail is delivered without excessive delay, at the expense of delaying junk mail.

Using data from two Internet facing mail servers, we show how it is possible to simply and accurately predict whether the next message sent from a particular server will be good or junk, by monitoring the types of messages previously sent. The prediction can be used to delay acceptance of junk mail, and prioritize good mail through the mail server, ensuring that loading is reduced and delays are low, even if the server is overloaded.

The paper includes a review of server-based anti-spam techniques, and an evaluation of these against the data. We develop and calibrate a model of mail server performance, and use it to predict the performance of the prioritization scheme. We also describe an implementation on a standard mail server.

1 Introduction

In recent years the volume of junk mail (spam, virus etc.) has increased dramatically. According to Message-Labs, the volume of spam increased by 77% in 2003, and virus-carrying emails increased by 84% [18]. This increase in volume taxes the resources of servers responsible for routing mail, and increases the transit time of emails. During virus/worm attacks these delays can increase to unacceptable levels [2]. This paper presents an approach that ensures that most of the “good” mail is processed quickly at the expense of larger delays for junk mail.

Unfortunately, this problem is only going to get worse. The volume of email traffic increases over time [33], but the volume of spam increases much faster. Message-Labs predict that by 2004 70% of all messages will be spam, as compared to (only) 55% as of November 2003 [17]. There are other reasons it is getting worse. First is the convergence of viruses and spam, where virus-infected machines are being sold and used as spam relays [35]. Secondly sites that offer blacklisting (lists of IP addresses used by spammers) have been subject to denial of service attacks [28], some launched from viruses [29]. Thirdly there is a continual arms race between spam and content filtering programs, with filtering always behind and missing some spam [20, 11].

Our approach is to enable good mail to reach the desktop without delay, as opposed to most existing approaches to the problem of spam that concentrate on preventing junk mail from reaching the desktop. Some of the stopping mechanisms reduce server load (and so reduce delays), for example the use of blacklists and other heuristics to refuse connections from spamming servers. Others increase server load, for example content checking and filtering at the server. As users are intolerant of any mistakes in classification (particularly false positives—legitimate mail classed as spam), and as spammers seek to evade the filters, spam checking becomes more computationally intensive, resulting in overloaded servers and increased delays.

Our aim is to provide good quality of service for “good” mail in spite of the volumes of junk. The idea is to not stop mail that is identified as junk from reaching users, but to delay it, by delaying its acceptance into the mail server and giving it low priority access to the content scanner or other bottlenecks in the mail system. This
means that false positives of our detection algorithm are not disastrous, because they do not prevent the (eventual) delivery of good email.

We present data collected from two heavily used Internet facing mail servers from a large corporation, and show that it is relatively simple to predict whether the next message from a particular server will be junk or not. The prediction is based on the types of messages (good/junk) that the server has previously sent, and is fairly accurate (approximately 74–80% of good messages and 93–95% of junk messages are recognized accurately). This prediction can be used to reduce server load by delaying acceptance of messages (using SMTP temporary failures [21]) and to prioritize messages through the mail server. This latter response allows good messages to be processed with minimal delay, even when the server is overloaded.

Using a model of the performance of a mail server, calibrated against our data, we show the effectiveness of this prioritization scheme. For a variety of loading conditions and server performances, the model predicts very low delays (small numbers of seconds) for good mail, compared to the (small numbers of hours) delays observed for a system without prioritization. This not only improves the quality of service, but also means that mail servers will no longer have to grow in processing power at the rate that junk mail increases.

The rest of the paper includes a review of server-based approaches to spam, and an evaluation of these responses given our data. The following sections then describe the prediction method, describe the performance model and present results for mail transit times. The final sections describe a practical implementation (currently under construction), and draw conclusions.

2 Related work

In the SMTP protocol [21] there is a defined point at which the receiving server takes responsibility for delivery of the message. This point is after all the data has been received and acknowledged. Responses to spam at the server can be divided into those before this point (pre-acceptance) and after (post-acceptance). Pre-acceptance responses are preferable, as by stopping the mail they reduce server load. Once the mail has been accepted the server is obliged to do work to deliver it.

Pre-acceptance responses can be divided into three types: blocking, delaying and tempfailing. Blocking means refusing to accept mail from a particular server, usually identified by IP address (because it is the only part of the SMTP opening dialog that is not easy to forge). Mail is not accepted if it is found on various lists or matches a number of heuristic rules designed to combat spam. For example, a Real-time Blacklist (RBL) is a maintained list of spamming IP addresses, provided by a variety of organizations, and generally accessed over DNS [27]. There are also lists of open relays (poorly configured or compromised machines on the Internet that relay indiscriminately), as well as dialup blocks. Other heuristics used include, for example, whether the sender’s domain is correct, and whether the server accepts incoming mails. Other blocks may be more receiver specific, for example Tantalus [9] blocks a server if it sends too many undeliverable messages.

Blocking is generally accepted and used, and while it is still vulnerable to false positives, there are well known mechanisms to deal with these (e.g.[16]). Unfortunately it takes time to add an address to a blacklist, during which the spammer can send many emails. In addition, our data will show that these blocks are only partially effective, mainly because it is easy for spammers to conform to the heuristics and change their IP address frequently.

The response of delaying is a more direct attack on the resources of the sender of junk mail. Teergrubing [8] is the practice of slowing down the SMTP conversation if a sender tries to send a message to too many recipients (or if the sender’s address is on a maintained list). This consumes spammers’ resources, but unfortunately also consumes resources on the server. There are many variants on this idea, e.g. SpamThrottle [36] which is a more sophisticated teergrubing algorithm, and various rate limiting mechanisms built into mail servers, e.g. rate limiting in Postfix [34]. Some delaying tactics work by analyzing the data in the message as it is loading, and slowing the connection if the message looks like spam (e.g. Tar Proxy [31]), and some are like honeypots for spam (e.g. Jackpot Mailsrverver [5], an open relay that talks slowly to spammers that attempt to use it).

Delivering is a form of rate limiting, and the above techniques limit the number of messages per SMTP connection, or the number of recipients per message. A rate limit on number of messages per sender is also possible, as offered by IronPort [13], who offer a rating of the trustworthiness of senders, so allowing servers to implement rate limits.

Most of these mechanisms assume that incoming spam
is detectable because it is sent at a high rate, and that slowing it down will reduce the amount of spam received. While it is possible for spam to arrive at a high rate e.g. during a dictionary attack [1, 14] where a spammer bombards a server with messages addressed to random email addresses, our data will show that this generally not the case. Most spam that is received by our servers is sent by servers from which only small numbers of messages are received, and those messages are not received at particularly high rates. This shows that rate-limiting mechanisms are unlikely to be effective, as the messages are already received at low rates.

The last pre-acceptance response is tempfailing. This is an SMTP control code that means “my server is temporarily unavailable, please try again later” to the sending server [21]. Well configured mail servers will retry the mail later, but currently most spamming software and email-based viruses do not attempt retries. This is the premise of Greylisting [12], where a server keeps a list of triples consisting of the sending host’s IP address, sender’s email address and recipient’s email address, and tempfails mail that would generate new triples for some period of time. Since spamming software does not retry, this vastly reduces the amount of spam accepted by a server running Greylisting, while normal mail is subject to some delay. The weakness of this approach is that if (and when) spammers implement retries, this technique will become less effective.

Tempfailing is a useful strategy when used in combination with blacklists. It forces the spammer to maintain its IP address for longer, and reduces the rate that the spammer can send mails. This reduces the number of mails that are sent before the address is added to a blacklist.

Although tempfailing is a useful tool in the fight against spam, our data shows that the Greylisting approach is rather inefficient, delaying a large proportion of good mail, and requiring rather large numbers of triples. We suggest that evidence of spamming can be better obtained by looking at the history of mail sent by a particular server, and that this combined with tempfailing can be effective.

The main post-acceptance response is to look at the content of the mail message, scan it for viruses and examine the text for evidence of spam. Spam filtering can be accomplished by a variety of means, including collections of simple rules [25], Bayesian reasoning [22], and collaborative filtering [6]. It can also be implemented on the client, but is increasingly implemented on the server, often as an add-on to existing scanning for viruses.

None of these filtering mechanisms are infallible, and all suffer from the problem of false positives or incorrectly classified mail. In addition, the filters must be continually updated, as spammers develop new means to evade them [11, 20]. In addition, scanning mechanisms cause extra loading at the server, resulting in delays when the server is heavily loaded.

In this paper we introduce a new post-acceptance response, that of prioritizing the flow of good mail through the mail server, and processing mail suspected of being junk (virus, spam, undeliverable) at reduced priority. The aim is that in times of heavy system load, there will be prompt delivery of good mail, at the expense of delays to other mail.

3 Characteristics of email traffic

To evaluate existing responses and develop new ones, we collected data from the logfiles of two bastion (Internet facing) email servers in a large corporation, as detailed in Table 1. Server1 and server2 are the primary and secondary server for a single email domain. These servers ran a virus checker (Sophos [26]), and a spam filter (Spam Assassin [25]), using MailScanner [32], so for each SMTP transaction it is possible to discover from the log files whether received mail contained viruses or was flagged as spam. Server1 and server2 did not store mail for reading, or check the addresses of recipients, but forwarded mail to other servers. Undeliverable mail was thus indicated by a failure to deliver to these servers, and was recorded in the logs. The logs also included records from incoming mail blocked because of real-time blacklists and other heuristics. This data gives a good picture of a server’s-eye view of spam.

These servers are listed in public MX lookups, but also receive some mail from other mail servers within the corporation (they are part of an internal mail routing chain). Because the intent of this work was to combat spam entering a corporation, we removed the data corresponding to this “indirect” mail, leaving only the “direct” messages—interactions with other mail servers over the Internet.

Both servers subscribed to real-time blacklists and performed other checks on the sender’s address before accepting messages. These mechanisms rejected around 34–36% of incoming messages (see Table 2). However, this is only partially effective at reducing spam, as around 65–70% of the accepted mail is junk, as shown
Table 1: Details of the data collected.

<table>
<thead>
<tr>
<th>Server</th>
<th>Length (Days)</th>
<th>Number of Messages</th>
<th>Number of Recipients</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server1</td>
<td>69</td>
<td>855228</td>
<td>1229459</td>
</tr>
<tr>
<td>Server2</td>
<td>69</td>
<td>755565</td>
<td>1097169</td>
</tr>
</tbody>
</table>

Table 2: Effectiveness of real-time blacklists and other blocking responses. While blacklists and other checks block around 34–36% of incoming messages, the accepted mail is still mostly junk, as shown in Table 3.

<table>
<thead>
<tr>
<th>Type</th>
<th>Server1</th>
<th>Server2</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBL</td>
<td>256700</td>
<td>207144</td>
</tr>
<tr>
<td>Open relay</td>
<td>119161</td>
<td>95536</td>
</tr>
<tr>
<td>Other checks</td>
<td>112555</td>
<td>81170</td>
</tr>
<tr>
<td>Total blocked</td>
<td>488416</td>
<td>383850</td>
</tr>
<tr>
<td>Total accepted</td>
<td>855228</td>
<td>755565</td>
</tr>
<tr>
<td>Total attempts</td>
<td>1344960</td>
<td>1139415</td>
</tr>
</tbody>
</table>

Table 3: Breakdown of mail messages by type for each server. The percentages for spam, undeliverable and virus do not sum to 100% as messages can fall into both types. The last two rows show totals of accepted good mail and accepted junk mail, where junk mail is any one of virus, spam or undeliverable.

<table>
<thead>
<tr>
<th>Type</th>
<th>Server1</th>
<th>Server2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good</td>
<td>260348</td>
<td>262941</td>
</tr>
<tr>
<td>Spam</td>
<td>497554</td>
<td>414234</td>
</tr>
<tr>
<td>Virus</td>
<td>2749</td>
<td>2371</td>
</tr>
<tr>
<td>Undeliverable</td>
<td>364487</td>
<td>298169</td>
</tr>
<tr>
<td>Total accepted good</td>
<td>260348</td>
<td>262941</td>
</tr>
<tr>
<td>Total accepted junk</td>
<td>594880</td>
<td>492624</td>
</tr>
</tbody>
</table>

Table 3 shows a breakdown of accepted messages. The values for spam and virus reflect messages that were flagged as such in the logs. A message was deemed undeliverable if on the first delivery attempt more than half of the recipients failed. The good messages are those that are left: not virus, spam or undeliverable. The table shows the enormous volume of mail that is junk (65–70% of all accepted mail). This is consistent with other measures of the prevalence of spam [15]. What is also surprising is the number of undeliverable messages. This appears to be mostly spam—of the 364,487 undeliverable messages for server1, only 94,735 (26%) were not classed as spam or virus. This is likely caused by “trolling” [12, 1], where spammers send messages to automatically-generated usernames at known domains, hoping that these correspond to genuine email addresses. This figure is far too high to be explained by mistyped addresses, corrupted mailing lists or servers that are offline. Assuming that nearly all of the undeliverable messages are spam suggests that a significant proportion are not being classified correctly by the spam filter. This in turn suggests that the measure for good in the table is optimistic: it includes some spam messages. Overall, these figures are alarming as they show how much resource is wasted passing junk emails through the email system.

Many of the pre-acceptance responses try to limit the rate that spam messages can be accepted by a server. However, our data shows that spam received tends to be at a low rate, and is mostly sent from servers that send small numbers of messages.

Table 4 shows the number of servers that send (only) good mail, (only) junk mail and a mixture, together with the number of messages sent. Only a small proportion of servers (11%) send only good mail, but are responsible for a small proportion of the messages (11%). By contrast, the overwhelming majority of servers send only junk mail (79%) but generate a relatively small proportion of the messages (48%), implying that each server sends few messages. The mixed class contains data for servers that sent at least one good and one junk mail. This includes those that send mostly good with the occasional junk (e.g. a mistyped address, a false positive spam detection), or those that send mostly junk with the occasional good (e.g. a spammer whose spam occasionally evades spam detection), or a more even mixture that might be obtained from an aggregating mail server (e.g. an ISP). This class is a small proportion of the servers, but generates a large proportion of the messages.

Figure 1 shows the same effect, but broken down by how
Table 4: Number of sending servers and the number of messages that they sent, classified into three groups: those sending only good mail, only junk mail and a mixture. Data taken from server1.

<table>
<thead>
<tr>
<th>type</th>
<th>servers</th>
<th>messages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>number</td>
<td>number</td>
</tr>
<tr>
<td>good</td>
<td>242407</td>
<td>97553</td>
</tr>
<tr>
<td>junk</td>
<td>178762</td>
<td>413,725</td>
</tr>
<tr>
<td>mixture</td>
<td>23416</td>
<td>343950</td>
</tr>
</tbody>
</table>

many messages each server has sent. The figure was constructed by counting how many messages of each type (good/junk) each server sent, and plotting the cumulative percentage of total messages against the number of messages per server. There are three lines, for good, junk and the total.

The line for junk mail shows that most of the junk mail comes from servers that send fewer than 100 messages. In fact 45% of junk mail comes from servers that send fewer than 10 messages. On the other hand, good mail is more likely to come from servers that send many messages. The total graph is a combination of these two effects, and is more influenced by the junk mail as that makes up the bulk of the messages.

These results suggest that the rate of incoming junk messages from each server is low (it has to be low if they send few messages—10% of junk mail comes from servers that send only one message, for which a rate limit is meaningless). Measurements of local frequency (number of mails per minute) show that some junk mail is sent at high rates, but this is a tiny proportion of the total mail (see Figure 2). There is very little difference between the rates of good and junk senders for the bulk of messages.

This result is perhaps surprising and counter-intuitive, particularly as many of the delaying responses are geared toward reducing rates. This data suggests that with the possible exception of dictionary attacks (which we did not notice in our data), rate-limiting mechanisms are not particularly effective.

The explanation for the shapes of Figures 1 and 2 is a combination of two factors. Firstly, spammers are forced to change their server addresses frequently as addresses are placed on blacklists, so the volume of mail from each spamming server is limited. Secondly, each mail server only sees a sample of the total spam mail sent by any spammer, corresponding to the number of messages on the spammer’s mailing list that are handled by the server. This further reduces the number of messages that are received from each server, and reduces the rate.

Even for good mail, a significant proportion comes from servers that send few messages. This has a bearing on the effectiveness of Greylisting [12], a technique that keeps a record of the triple \{sender IP address, sender email address, recipient email address\} and only allows mail through without delay if that triple has been observed before. Table 5 shows an evaluation of Greylisting. These results are for 36 days worth of data, the standard lifetime of a triple. The percentage of good mail that is delayed is rather large (40–51%). On the other hand, Greylisting is very effective against junk mail, with 98% being delayed. The maximum number of triples is enormous, but the actual number required to be stored is likely to be a lot lower—Greylisting only keeps triples that have accepted mail.

However, the fundamental idea behind Greylisting, that unusual mail is likely to be junk, is a good one, and we explore this further in the following section.
Figure 2: Cumulative histogram of incoming message rate calculated on a per sender basis and normalized. The graph shows that the message rates for good (*) and junk (+) mail are very similar. In addition, most of the mail is sent at a very low rate (96% of servers send messages at 1 message/minute). There are some junk mail senders that send messages at higher rates (40–100 messages/minute), but they are a tiny proportion. Since most servers send small numbers of messages (Figure 1), this figure shows that the vast proportion of junk mail is received at a low rate, and that the rates for junk and good mail senders are indistinguishable.

Table 5: Results for performance of Greylisting. Greylisting is effective against junk mail, with 98% being delayed, but less effective with good mail, with 40–51% being delayed.

<table>
<thead>
<tr>
<th>type</th>
<th>number</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>server1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>triples</td>
<td>619536</td>
<td></td>
</tr>
<tr>
<td>good delayed</td>
<td>84748</td>
<td>51% of good</td>
</tr>
<tr>
<td>junk delayed</td>
<td>444829</td>
<td>98% of junk</td>
</tr>
<tr>
<td>server2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>triples</td>
<td>557314</td>
<td></td>
</tr>
<tr>
<td>good delayed</td>
<td>69227</td>
<td>40% of good</td>
</tr>
<tr>
<td>junk delayed</td>
<td>376532</td>
<td>98% of junk</td>
</tr>
</tbody>
</table>

4 Using sending history to predict future mailing behavior

This section develops the idea of using the past history of types of mail sent (good/junk) in order to predict the type of the next message from a server before that mail is accepted. This enables new and interesting responses both pre- and post-acceptance.

The data in the previous section (particularly Table 4) shows that mail servers can be classified based on the mail that they have sent, into those that send junk mails, good mails and a mixture. This suggests that looking at history might be a good way to predict behavior.

One simple way to represent history is to calculate the proportion of good messages received by the server, where a good message is one that is not classed as spam or virus, and one where more than half of the recipients are delivered on the first delivery attempt. The proportion of good mail $P_i$ can be calculated as

$$P_i = \frac{N_{good}(i)}{N_{total}(i)}$$

where $i$ is the server indexed by IP address, and $N_{good}$ and $N_{total}$ are the number of good and total messages received from server $i$. When the first message is received, there is no data to calculate $P_i$, so it is initialized to zero. As messages are scanned and delivery attempts are made, the values of $N_{good}$ and $N_{total}$ are updated. $P_i$ can be used to predict whether a server is likely to send good or junk mail using a simple threshold $r$, i.e. if $P_i > r$ the message is predicted to be good and vice versa. As more messages are received, this prediction becomes more accurate (see Figure 3).

Table 6 shows the overall accuracy of this method when $r = 0.5$. It shows that it is important to include all forms of junk mail in order to get an accurate classification; just basing the calculation on spam alone does not give such accurate results. This is most likely due to the fact that by combining spam detection and undeliverable mail combines two noise indicators of spam to give greater accuracy: a proportion of spam evades the spam filter, and many of the addresses on spam mailing lists are incorrect.

Table 6 shows that this measure is remarkably accurate at detecting junk mail (93–95% accuracy), and also good at detecting good mail (74–80%). Some of the errors are unavoidable, as the first mail from every sending
server will be predicted to be junk. Although most of the mail sent by new servers is indeed junk, this results in a small number of good messages being misclassified. Out of 855228 messages on server 1, 226585 (26%) were the first message from a new server, and of those 33021 (15%) were good. The other cases where good messages are classed as junk appear from a cursory look to mostly be spam that has evaded the content scanner (for example they have implausible sender email addresses). Thus the errors may be less than measured.

The other type of error, where junk messages are classed as good, occurs for an even smaller group. It is probably caused by mistyped addresses, cases where the spam detector over-zealously marks a legitimate mail as spam, or where a sender that has previously sent good mail gets infected by a virus.

This method is practical to calculate on a real mail server. It requires some state \((N_{\text{good}}, N_{\text{total}})\) to be maintained for each server, but that state is far smaller than that required for e.g. Greylisting. In addition, that state can be updated asynchronously as it becomes available. In a real implementation, this information will be delayed, however this is unlikely to affect the accuracy of the results significantly (it will only affect updates during overloading).

The results in Table 6 used the entire 69-day dataset to calculate \(P_t\), however the probability is remarkably accurate even if only a small part of the history is maintained. Figure 3 shows how even with a very short history of a few messages, the probability measure performs well. This is important as it will quickly capture changes in server behavior, for example if a previously good server starts sending mail infected by a virus. In addition, if no mail has been sent from a particular server for a long time, the record for the server’s IP address can be safely removed without noticeably affecting performance.

This method of predicting good and junk mail is powerful because it can be calculated before the message is received. It thus enables both pre- and post-acceptance responses.

The rate of false positives is too high to use \(P_t\) for blocking mail (in fact if it were used, no mail would ever be delivered because the first mail from every server would be rejected!). It can however be used with tempfailing. As mentioned above, the advantage of tempfailing is that it effectively blocks mail if the spammer does not retry, and increases the efficiency of blacklists if they do. An appropriate use of tempfailing with the sending history measure is for example to not accept any mail from a new server for a short period of time (e.g. 4 hours), and
from predicted junk-sending servers for a longer period (e.g. 12 hours). Tempfailing could also be used when the server is heavily loaded, preferentially tempfailing junk mail.

The history measure also enables an exciting post-acceptance response, that of prioritizing good mail through the bottlenecks in the server (often the virus/spam scanner). Instead of all mail being treated equally, good mail can be placed in a high priority queue for the scanner and SMTP processes. This will enable good mail to travel through the server without delay, even when the server is very heavily loaded by spam or virus attack (both of which are increasingly common).

5 Testing the responses

This section provides some evidence for how much effect the responses described above would have on reducing the amount of spam processed, and reducing the effect of the volume of junk mail on the flow of good mail through a mail server.

Firstly, we consider the effect of tempfailing new servers and servers predicted to send junk. Unfortunately it is very difficult to test this just using log data, as the response is to request the remote sending server to retry later, and it is difficult to predict the behavior of the sending server.

A best-case estimate would be to assume that spammers and virus-infected machines do not retry, but that good senders do. Thus out of 855,228 messages accepted by server 1, if a 4 hour delay was used for new senders and a 12 hour delay for junk mail, the effect would be as in Table 7. Only 26% of good mail would be delayed, and as discussed above, a proportion of this mail is likely to be misclassified junk. If this sort of system were widely deployed, it is likely that spammers would implement retries. This would cause the amount of junk mail rejected to decline considerably.

It is much easier to predict the effect of post-acceptance responses. We do this by constructing a model of a mail server that allows us to calculate the time taken to process a mail message, under different amounts of loading. We can then alter that model to incorporate prioritization schemes and predict the final performance of the system.

The initial model of the system is shown in Figure 4 (a). This is a generic model of a mail server that includes a mail scanner or filter. The incoming mail is handled by an SMTP process, before being scanned for viruses and spam by the mail scanner, and being delivered using a second SMTP process. (b) Model of mail server with prioritization. The incoming process places mail in one of two queues depending on the predicted message type. The mail scanner selects messages from the queues using a scheduling algorithm.

<table>
<thead>
<tr>
<th>type</th>
<th>number</th>
<th>percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>good delayed by 4 hours</td>
<td>33021</td>
<td>13% of good</td>
</tr>
<tr>
<td>good delayed by 12 hours</td>
<td>34899</td>
<td>13% of good</td>
</tr>
<tr>
<td>junk mails rejected</td>
<td>565922</td>
<td>95% of junk</td>
</tr>
</tbody>
</table>

Table 7: Effect of pre-acceptance responses on good and junk mail. Data from server1.

![Figure 4: (a) Model of basic mail server. Incoming mail is handled by an SMTP process, before being scanned for viruses and spam by the mail scanner, and being delivered using a second SMTP process. (b) Model of mail server with prioritization. The incoming process places mail in one of two queues depending on the predicted message type. The mail scanner selects messages from the queues using a scheduling algorithm.](image-url)
simple scheduling algorithm to select which message to process next. The simplest algorithm is “absolute priority”, where the scanner always takes from the high priority queue unless it is empty, when it services the low priority queue [19]. The scheduling operation is modeled as taking \( t_{\text{schedule}} \). As before the total time is the sum of the individual times plus the time spent queuing.

We simulated both models using the DEMOS system performance modeling tool [4]. This allows synthetic traffic to be “played” through the model and the system performance evaluated. The parameters of the model (the various service times \( t_{IN}, t_{SCAN}, t_{OUT} \)) were estimated from the log data. These parameters are difficult to estimate, because the times in the logs include messages being queued as well as processed. To deal with this we took a lower bound on the actual measurements, assuming that the fastest times corresponded to small or no queues. In addition, we used a facility within DEMOS to specify a probability distribution of parameters rather than a fixed value. The most important parameter is the time to scan a message. This turned out (somewhat surprisingly) to not be particularly sensitive to message size, and was modeled as an exponential distribution. All the other aspects of the model were taken from the log data: the rate and ratio of good/junk were taken from incoming message rates with different rates for day and night; and the traffic was assigned to different priority queues using the probabilities in Table 6, i.e. a good message had a 74% chance of being placed in the high priority queue.

It is important with any model to calibrate it accurately, so that it is possible to believe its predictions. Figure 5 shows the distribution of delays for server1 together with the delays predicted by the model, with parameters as given in the caption. The figure shows that the model fits the data reasonably well.

Having calibrated the model, the effect of the prioritization can be tested. Figure 6 shows the effect of the same parameters applied to the model in Figure 4. The extra processing times \( t_{\text{classify}} \) and \( t_{\text{schedule}} \) were assumed to be small compared to the scan time \( t_{\text{scan}} \) and were neglected. The plot shows the average delay against the rate of incoming traffic. For low traffic loads the delays are small, because the server is lightly loaded. As the load increases, if no prioritization is used, the delays increase sharply. If prioritization is used, the good mail continues to be processed with small delays, but the junk mail is heavily delayed. These traffic loads are common in practice: marked on the diagram are normal loads and loads sampled during the SoBig.F virus attack [30]. For example, during the virus attack, using a conventional arrangement, the mail was delayed by 2.7 hours. With the prioritization scheme, good mail placed in the high priority queue is delayed by only 22 seconds on average, with junk mail delayed by over 4 hours.

Different mail servers have different performance capabilities (different hardware/software configurations), which affect their ability to process mail. Figure 7 shows the behavior of the system with a constant traffic load, but varying server performance. When the server is over-provisioned the delays are small, but for an under-powered server good mail is still passed with very small delays. This means that mail servers can be sized based on expected normal mail loading, and that their performance will be less sensitive to the amount of junk mail processed. This is a great benefit given the volumes of spam and virus mail that is processed now as well as the volumes being predicted [18].

Of course, some good mail is wrongly classified as junk, and will have delays similar to the junk mail. However, this delay is only slightly worse than that experienced in the original system for all mail, only a small proportion of mail is affected, and some of that mail should have been classed as spam anyway. In any case it should only be the first legitimate mail from any server that will be
Figure 6: Average delay against traffic loading for the original model (○) and for the good (×) and junk (+) queues when prioritization is used. The (*) line is the average delay for good mail (some of which is processed in the good queue, some in the junk queue). The two vertical lines indicate traffic loading which is normal, and loading observed during the SoBig.F virus attack. When the load increases (smaller time between messages), good mail does not get delayed, but junk mail is delayed more than if there were a single queue. Some good mail is placed in the junk queue, but overall the average delay for good mail is around half what it would be with a single queue.

Figure 7: Average delay against server performance ($t_{SCAN}$) for the original model (○), good queue (×), junk queue (+) and average good (*). The top plot corresponds to normal and the lower to SoBig traffic loading. With a single queue, a highly rated server is needed to provide good performance under heavy loading, while with prioritization good mail has very low delays even for quite slow servers.
Mail is interrupted using the milter interface, to query the ProbabilityCalculator for each connection whether to tempfail, or how to mark the message. Mail is placed in the spool $S$ as usual, and moved into the appropriate queue by the MailMover process. The copied MailScanner then processes the two queues. Information about the message type is fed back from log files via the Log-Monitor process.

6 Implementation

Figure 8 shows a sketch of how these schemes could be implemented on an industrial strength mail server. Sendmail [24, 23] and MailScanner [32] are used as examples, but the approach can be generalized to other products.

The basic idea is to mark messages as good or junk in the incoming sendmail process, and dump them in the usual spool directory. The messages are then moved into two queues, which are serviced by two copies of the mail scanner.

When mail is received by the incoming sendmail process, it can be intercepted using the Milter interface [37]. This interface allows access to the mail message and header information, and also allows certain actions like tempfailing or blocking, as well as writing headers and changing the message body. The specific milter code for this system would grab the IP address of the sending server and use that to query the “ProbabilityCalculator”. The ProbabilityCalculator keeps the history information for each sender IP address (e.g. $N_{good}$/$N_{total}$), and uses that to determine the correct action. This action could be to tempfail the request (the ProbabilityCalculator will keep track of when messages will be accepted from particular servers), or to mark the message as good or junk, for example by writing an extra header into the mail message. If the message is accepted it is written into the usual mail spool $S$.

The “MailMover” component runs regularly, scanning messages in the spool to determine whether they are good or junk, and moving them into the two queues $HI$ and $LO$ respectively. Sendmail simply writes the messages into the spool, so it is safe to move the files (other servers e.g. postfix [34] and qmail [3] rely on more complex file information and a more sophisticated system would be needed). Two copies of the MailScanner then process the two queues. These programs would be configured so that the one servicing the high priority queue ran faster than the other.

MailScanner automatically triggers the outgoing SMTP process when it has processed a (configurable) number of messages. Any that are undeliverable are saved in a separate queue and another sendmail process handles retries. It would be possible to implement prioritization for this process too, using another MailMover and two retry sendmail processes.

Once the message has been processed by the scanner, information about that message is available to update the history of the sending server. In addition, once the outgoing sendmail process has attempted to deliver the message, information about undeliverable messages is also available. Since this information is written into the normal syslog, a process that monitors the logs (“Log-Monitor” on Figure 8) can recover this information and update the history in the ProbabilityCalculator. It does this by keeping track of message identifiers and matching the results of the scan/attempted delivery with the IP address of the original sending server.

The ProbabilityCalculator is shown here on a single system, however there is no reason why the information stored should not be shared by many servers. Sharing
information is likely to further improve the accuracy of the classification.

This system is currently under development and we expect testing to begin shortly.

7 Conclusion

This paper has considered the problem of unacceptable delays in sending email due to servers clogged with unwanted email messages—viruses, spam and undeliverable mail. This is a problem that is significant now, and is likely to become more significant as the volumes of spam and virus-carrying email increase.

We have shown, with reference to empirical data, that existing mechanisms to deal with spam (blacklists, rate limiting, greylisting, content filtering) are only partially effective. So, we have developed a system to preserve performance while coping with large volumes of junk mail.

We have shown that a simple prediction of the type (good/junk) of the next message received by a server can be used to delay acceptance of junk mail, and to prioritize good mail through the bottlenecks of the server. The prioritization scheme ensures that most of the good mail is transmitted with small delays, at the expense of longer delays for junk mail. This scheme greatly improves on the performance of current non-prioritized schemes.

We have also argued that this approach is practical, and sketched an implementation on an industry standard mail server that requires no modifications to existing code.

While this approach does not stop junk mail, it should increase the resilience of the mail system, making it better able to cope with overloading from spam and from virus attacks.

Acknowledgements

The authors would like to thank John Hawkes-Reed and Andris Kalnozols for help with data collection and developing the implementation.

References


