Email Prioritization: reducing delays on legitimate mail caused by junk mail[±]

Richard Daniel Twining, Matthew M. Williamson, Miranda Mowbray
Maher Rahmouni
Digital Media Systems Laboratory
HP Laboratories Bristol
HPL-2004-5(R.1)
May 24, 2004*

email, spam, junk

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.

* Internal Accession Date Only
[±] Proceedings of Usenix '04: General Track, June 30-July 2, 2004, Boston, MA, USA

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@dantwining.com, matthew_williamson@alum.mit.edu, {miranda.mowbray, maher.rahmouni}@hp.com

April 26, 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 MessageLabs, 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 [34], but the volume of spam increases much faster. MessageLabs predict that in 2004 70% of all messages will be spam, as compared to (only) 55% as of November 2003 [17]. There are other reasons why it is getting worse. First is the convergence of viruses and spam, where virus-infected machines are being sold and used as spam relays [37]. Secondly, sites that offer blacklists (lists of IP addresses used by spammers) have been subject to denial of service attacks [28], some launched from viruses [30]. 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, in contrast to most existing approaches to the problem of spam, which concentrate on preventing junk mail from reaching the desktop. Some of the prevention 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 not to 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, present initial results, 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 mail indiscriminately), as well as dialup blocks. Other heuristics used include whether the sender's domain is correct, whether the server accepts incoming mails, or whether the sender's address matches the sending domain [29]. 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. more sophisticated teergrubing algorithms (SpamThrottle [38]), and various rate limiting mechanisms built into mail servers, e.g. rate limiting in Postfix [35]. 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 [32]), and some are like honeypots for spam (e.g. Jackpot Mailswerver [5], an open relay that talks slowly to spammers that attempt to use it).

Delaying 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 the 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 is 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.

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], in which 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 the 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]. Moreover 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, or 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 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 [33], 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

Table 1: Details of the data collected.

server	length (days)	number of messages	number of recipients
server1	69	855,228	1,229,459
server2	69	755,565	1,097,169

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.

type	server1		server2	
	number	%	number	%
rbl	256,700	19	207,144	18
open relay	119,161	9	95,536	8
other checks	112,555	8	81,170	7
total blocked	488,416	36	383,850	34
total accepted	855,228	64	755,565	66
total attempts	1,344,960	100	1,139,415	100

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 multiple 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.

type	server1		server2	
	number	%	number	%
good	260,348	30	262,941	35
spam	497,554	58	414,234	55
virus	2,749	0.3	2,371	0.3
undeliverable	364,487	43	298,169	39
total accepted good	260,348	30	262,941	35
total accepted junk	594,880	70	492,624	65

around 65–70% of the accepted mail is junk, as shown in Table 3.

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. These appear to be mostly spam—of the 364,487 undeliverable messages for server1, only 94,735 (26%) were not classed as spam or virus. A likely cause is "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 many resources are wasted in passing junk emails through the email system.

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. A small proportion of servers (11%) send only good mail, and these are responsible for a similar 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 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 num-

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.

type	servers		messages	
	number	%	number	%
server1				
good	24,407	11	97,553	11
junk	178,762	79	413,725	48
mixture	23,416	10	343,950	40
server2				
good	20,508	10	80,082	11
junk	157,467	80	344,669	46
mixture	18,390	9	330,814	44

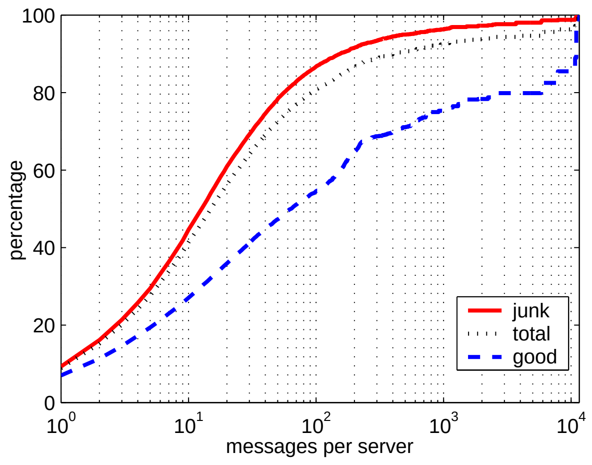


Figure 1: Cumulative histogram of number of messages received from each connecting server, subdivided into number of good, junk, and total. Junk mail tends to be sent by servers sending small numbers of messages (e.g. 50% of junk mail is sent by servers sending fewer than 20 messages). Good mail also has a high proportion from servers sending small numbers of messages, but the effect of large senders is more significant (50% of good mail is sent by servers sending fewer than 60 messages). The data is taken from server1.

ber 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" line is a combination of these two effects, and is more influenced by the junk mail as that makes up the bulk of the messages.

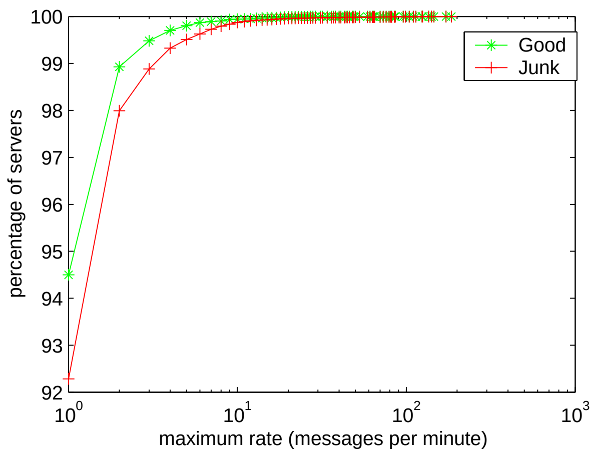
These results suggest that the rate of incoming junk messages from each server is low (it has to be low if they send few messages—10% of junk mail comes from servers that send only one message, for which a rate limit is meaningless). Measurements of local frequency (number of mails per minute) show that 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 against junk mail.

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. For servers that handle mail for large domains (e.g. Yahoo) this second factor will be less significant.

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 maximum number of triples is enormous, but the actual number required to be stored is likely to be a lot lower—Greylisting only keeps a triple for 36 days if a mail with that triple is accepted. Greylisting is very effective against junk mail, with 98% being delayed. On the other hand, the percentage of good mail that is delayed is rather large (40–51%).

However, the fundamental idea behind Greylisting—that



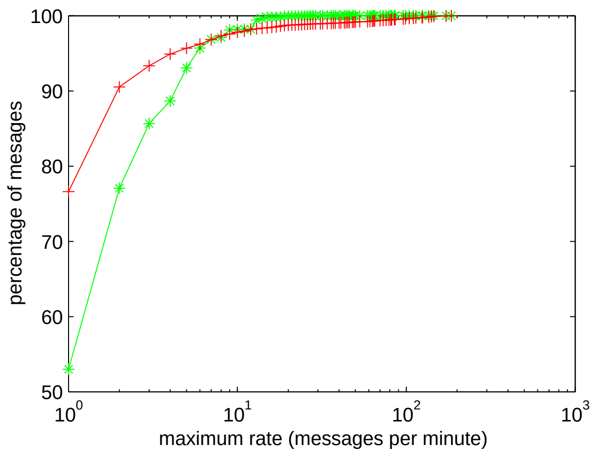


Figure 2: Incoming message rates. The top plot shows the cumulative histogram of incoming message rate, calculated on a per sender basis and normalized. The graph shows that the message rates for good (*) and junk (+) mail are very similar, and most of the mail is sent at a very low rate (92–94% of servers send messages at 1 message/minute). The lower plot shows the cumulative histogram of messages, calculated by assuming that each server sends all its messages at its maximum rate. This is therefore a worst case estimate of the proportion of mail sent at different rates. The bulk of both junk and good messages are sent at rates of less than 10 messages/minute.

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

type	number	%
server1		
triples	619,536	
good delayed	84,748	51% of good
junk delayed	444,829	98% of junk
server2		
triples	557,314	
good delayed	69,227	40% of good
junk delayed	376,532	98% of junk

unusual mail is likely to be junk—is a good one, and we explore this further in the following section.

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

$$P_i = \frac{N_{good}(i)}{N_{total}(i)} \qquad (1)$$

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

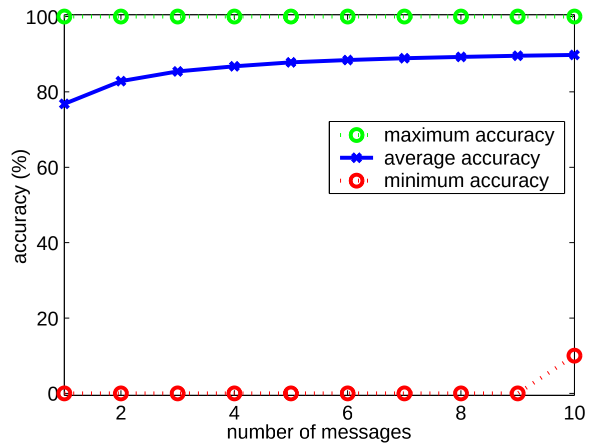


Figure 3: Plot showing the maximum, minimum and average accuracy of the prediction calculated using different numbers of messages, with $r = 0.5$. This data is for all servers that send at least 10 messages. The best case has all predictions correct, the worst 1 correct out of 10. The average goes quickly to near 90%. This result highlights that accuracy improves with history (from 77% with one message to 90% with 10), and also that not much history (< 10 messages) is needed to accurately predict message type.

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 combining spam detection and undeliverable mail combines two noisy 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

Table 6: Accuracy of classification when different data is taken into account. This was calculated by running through the logs and for each sending server evaluating P_i, classifying the message, and updating P_i ready for the next message. Each line in the table includes the effect from the lines above, i.e. the line marked undeliverable is performance for spam + virus + undeliverable. The classification is best when all the types of junk mail are taken into account.

	good		junk	
	number	%	number	%
server1				
spam	242,163	68	462,516	93
virus	239,990	67	464,709	93
undeliverable	192,428	74	565,922	95
server2				
spam	250,496	73	377,212	91
virus	248,516	73	379,112	91
undeliverable	209,309	80	459,432	93

855,228 messages on server1, 226,585 (26%) were the first message from a new server, and of those 33,021 (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 fewer 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.

Figure 4 shows that the prediction accuracy is insensitive to the value of the threshold r. Varying r in the range 0.1–0.8 gives around 10% variation in the individual accuracies, and virtually no variation in overall accuracy.

The results in Table 6 used the entire 69-day dataset to calculate P_i, however the probability is still 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 allow the quick capture of changes in server behavior, for example if a previously good server starts sending mail infected by a virus.

In addition, good performance can be achieved without

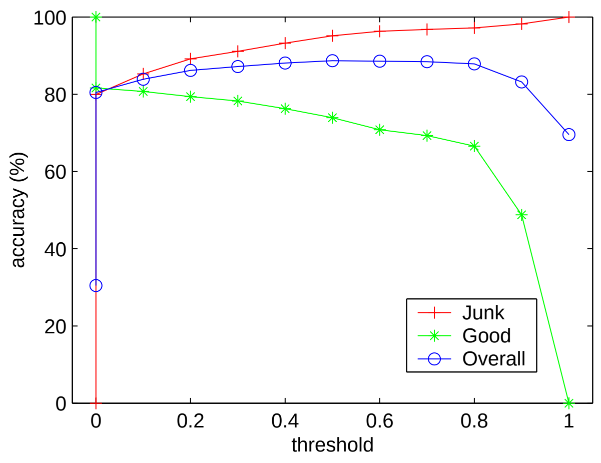


Figure 4: Plot showing the classification accuracy for different values of the threshold r. The overall accuracy is calculated by dividing the total number correct by the total number of messages. A wide range of values of r give good performance. It is interesting that even with $r = 0$, there is a reasonable prediction accuracy. This is because a sender is only classified as good if $N_{good}/N_{total} > r$. For 100% accuracy for good (0% for junk) r needs to be less than zero.

maintaining a record for every sending server. Figure 5 shows the effect on accuracy of maintaining a fixed number of IP records, using a first-in-first-out replacement policy. The total number of unique senders observed is indicated by the vertical line. It is only when the number of addresses stored is around a quarter of the total that the performance starts to fall off.

To summarize, this prediction method is practical to calculate on a real mail server. It requires some state (N_{good}, N_{total}) to be maintained for each server, but that state is either a pair of numbers, or a short (< 10) history of message types. In addition, state can be maintained for a relatively small proportion of the sending servers without sacrificing performance. The amount of state required is far smaller than that required for techniques like Greylisting. The state can also 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).

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

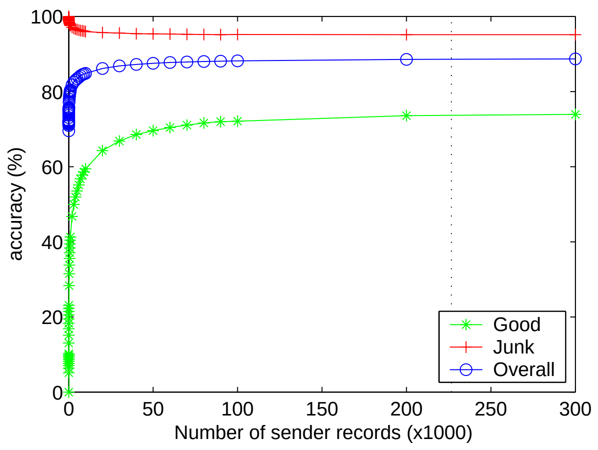


Figure 5: Prediction accuracy for $r = 0.5$ as a function of the number of sender history records maintained, with the vertical line indicating the total number of unique senders in the 69-day dataset.

The rate of false positives is too high to use P_i 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 spammers do not retry, and increases the efficiency of blacklists if they do. An example of an appropriate use of tempfailing with the sending history measure is 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). This technique would help combat email storms caused by mass mailing viruses: when a previously good machine starts sending the virus, its value of P_i will decrease and eventually its traffic will be tempfailed, so reducing the load on the server. There are, however, other more direct mechanisms to deal with email storms, e.g. [36]. Finally, 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. The good mail is still scanned, but will travel through the server with minimal delay, even when the server is very heavily loaded by spam or virus attack (both of which are increasingly common).

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

type	number	percentage
good delayed by 4 hours	33,021	13% of good
good delayed by 12 hours	34,899	13% of good
junk mails rejected	565,922	95% of junk

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 on 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 predicted junk mail, the effect would be as in Table 7. Only 26% of good mail would be delayed, and as discussed above, a significant 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 6 (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 that writes the mail to a local disk or spool q_{MS}, taking time t_{IN}. The mail is then loaded, scanned and placed in a second spool q_{OUT} by the mail scanner, marking the mail accordingly (normally by writing a header), and taking on average t_{SCAN}. The mail is then taken from this second spool and delivered

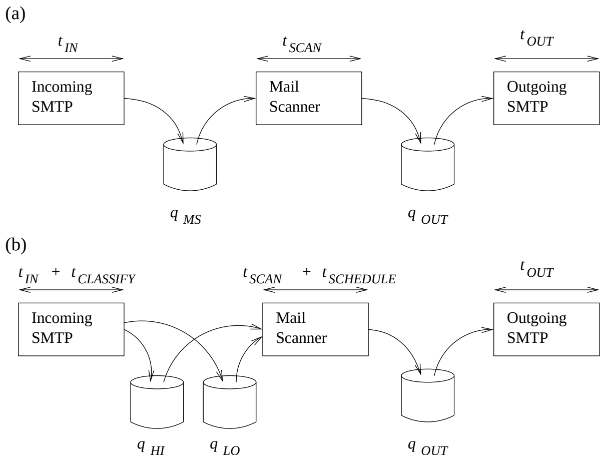


Figure 6: (a) Model of basic mail server. Incoming mail is handled by an SMTP process, before being scanned for viruses and spam by the mail scanner, and being delivered using a second SMTP process. (b) Model of mail server with prioritization. The incoming process places mail in one of two queues depending on the predicted message type. The mail scanner selects messages from the queues using a scheduling algorithm.

by the outgoing SMTP process, taking t_{OUT}. The overall time will be $t_{IN} + t_{SCAN} + t_{OUT}$ plus the time that mails spend on the queues waiting to be processed. As each of the spools is effectively a queue, this system can be analyzed using Queueing Theory [10].

Prioritization can be implemented by having two queues for the mail scanner, one for high priority or good mail (q_{HI}) and the other for low priority or junk mail (q_{LO}). When a mail arrives it is classified into good or junk (this extra processing taking $t_{CLASSIFY}$), and placed into one of these queues. The mail scanner then uses a 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 modelled as taking $t_{SCHEDULE}$. 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 modelling 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 mes-

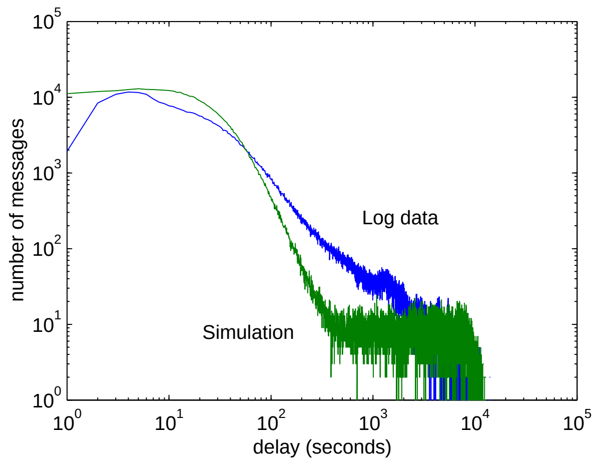


Figure 7: Histogram of message delays. The simulation captures the rough shape of the data, with the bulk of the messages delivered promptly, and a long "tail" of delays. The three service times were modelled as exponential distributions with means of $t_{IN} = 0.02s, t_{SCAN} = 7.9s, t_{OUT} = 0.08s$. The incoming message rate was modelled as a exponential distribution with mean $7.37s$ during the day and $11.08s$ at night.

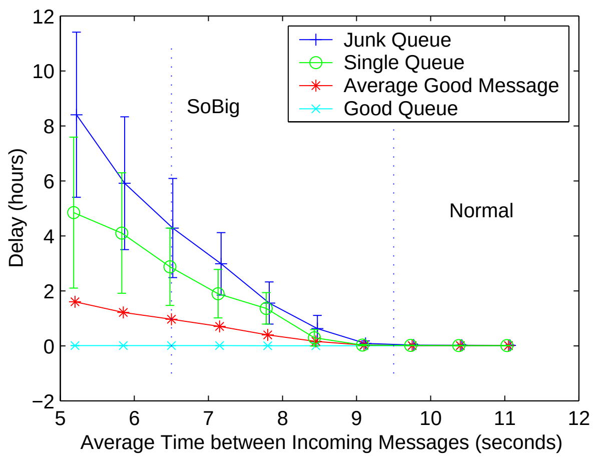


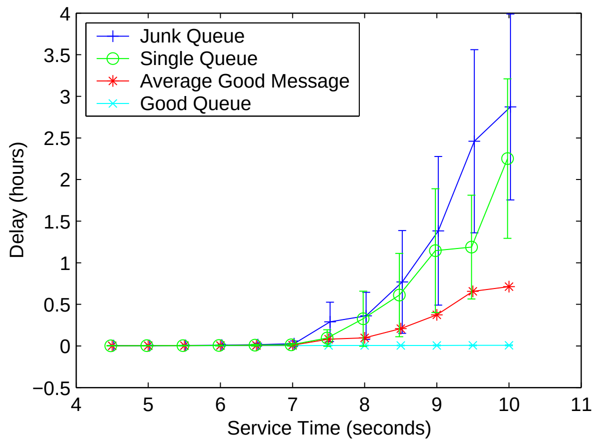
Figure 8: Average delay against traffic loading for the original model (○), the good (×) and the junk (+) queue, with error bars showing standard deviations. The two vertical lines indicate traffic loading that is normal, and loading observed during the SoBig.F virus attack. The average delay for good messages (*) is also shown, but is not particularly meaningful. Good messages that go in the good queue have small delays, but those wrongly placed in the junk queue will have delays similar to real junk messages. The distribution of delays for good mail is thus bimodal, and not easy to represent with means and standard deviations, especially as the differences in the two distributions are so large, e.g. 6 hours for junk and ≈20 seconds for good at 6s between messages.

sages 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) not to be particularly sensitive to message size, and was modelled 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 an 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 7 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 8 shows the effect of the same parameters applied to the model in Figure 6. The extra processing times $t_{CLASSIFY}$ and $t_{SCHEDULE}$ were assumed to be small compared to the scan time t_{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, the delays increase sharply if no prioritization is used. 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 [31]. 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 9 shows the behavior of the system with a constant traffic load



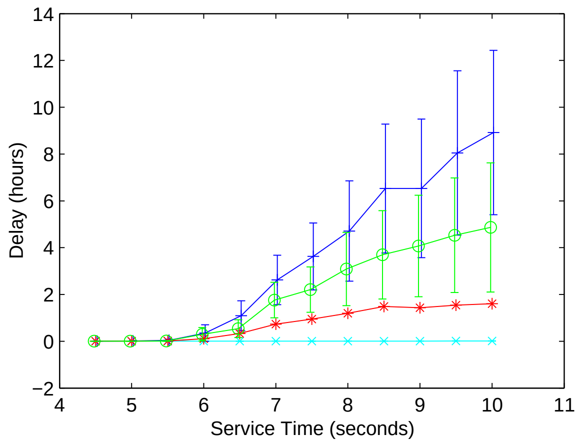


Figure 9: Average delay against server performance (t_{SCAN}) for the original model (∘), good queue (×), junk queue (+) and average good (*). The top plot corresponds to normal and the lower to SoBig traffic loading. With a single queue, a highly rated server is needed to provide good performance under heavy loading, while with prioritization good mail has consistently low delays even for quite slow servers.

but varying server performance. When the server is over-provisioned the delays are small, but for an underpowered 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, let alone 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 classified as spam anyway. In any case it should only be the first legitimate mail from any server that will be heavily delayed.

There are different scheduling schemes that can be used, for example absolute priority, fair share or weighted fair share [7]. We modelled several, but found that the choice of scheduling scheme has little effect on the delivery of good mail: the main effect is on the length of delays to junk mail, with absolute priority giving the longest delays.

To summarize, adding prioritization is extremely effective at ensuring that the bulk of good email is delivered promptly, even when the mail server is very heavily burdened by junk mail.

6 Implementation

Figure 10 shows a sketch of how these schemes could be implemented on an industrial strength mail server. Sendmail [24, 23] and MailScanner [33] 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 [39]. 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

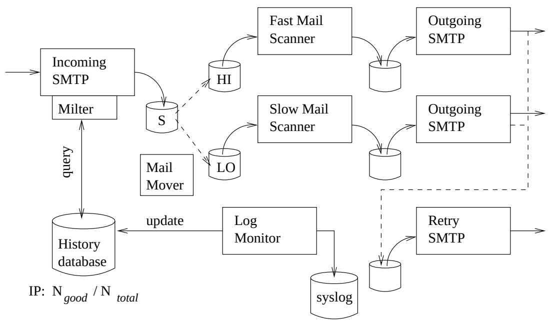


Figure 10: Schematic of implementation on sendmail. Mail is interrupted using the milter interface, and the milter queries the history database to determine for each connection whether to tempfail, or how to mark the message. Mail is placed in the spool S as usual, and moved into the appropriate queue 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 LogMonitor process.

changing the message body. The specific milter code for this system would grab the IP address of the sending server, and query the history database to determine the probability that the message was good or junk. The milter would then implement the action required: either to tempfail the request (writing to the database the time when messages from that server would next be accepted), or marking 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 [35] and qmail [3], rely on more complex file information; for these 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 scanner has processed the message, information about the 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 ("LogMonitor" in Figure 10) can recover this information and update the history in the database. 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 database 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.

7 Preliminary Testing

The system described above was implemented on a linux machine (1.6GHz RedHat 9.1, Sendmail 8.12.11, MailScanner 4.26.28, MySQL 3.23.58). The Milter, MailMover and LogMonitor were all implemented as separate perl processes.

For testing purposes two otherwise identical machines were used, one configured to use the prioritization scheme, and another configured to use a single MailScanner. Figure 11 shows the results of sending the same high load of messages to both machines. The figure clearly shows low delays for good messages even when the load is large.

8 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

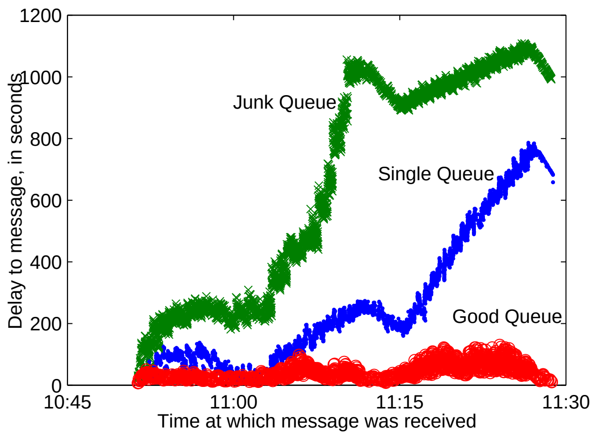


Figure 11: Delays for messages sent through a single queue machine, and the good and junk mail sent through a machine with prioritization. The prioritized good mail is delivered more promptly than it would have been without prioritization.

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. Initial tests of this implementation suggest that it works well in practice.

Not all spam classification occurs at the server, and the spam classification at the desktop is often more customizable and accurate. It would be useful to be able to feed this information back to the server. The challenge would be how to achieve this in a practical and secure way. One possible technique would be to write the originating server's IP address in the mail header and allow client software to "update" the mail server with more accurate spam classifications.

In conclusion, 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

[1] 4RSG LLC. What is a dictionary attack? Available at `http://www.filterpoint.com/help/dictionary.html`, 2003.

[2] Garry Barker. Spam surge slows email. Available at `http://www.theage.com.au/articles/2003/10/13/1065917342993.html`, October 2003. The Age, Technology Section, October 14 2003.

[3] Dan Bernstein. Qmail. Available from `http://cr.yp.to/qmail.html`, 1997–1998.

[4] Graham Birtwistle and Chris Tofts. DEMOS. Available from `http://www.demos2k.org/demos_ovr.htm`, 1979–2002.

[5] Jack Cleaver. Jackpot Mailswerver. Home page: `http://jackpot.uk.net`, January 2003.

[6] Cloudmark. SpamNet: Learn more: Collaborative SpamFighting. Available at `http://www.cloudmark.com/products/spamnet/learnmore/`, 2003.

[7] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing alorithm. In *Proceedings of the ACM SIGCOMM '89 Symposium*, pages 1–12, December 1989.

[8] Lutz Donnerhacke. Teergrubing FAQ. Available at `http://www.iks-jena.de/mitarb/lutz/usenet/teergrube.en.html`, 1996–2003.

[9] Brian Gannon. Tantalus v 0.02 – Perl Anti-SPAM Milter. Available at `http://www.linuxmailmanager.com/tantalus.html`, 2003.

[10] Donald Gross and Carl M. Harris. *Fundamentals of Queueing Theory*. Wiley–Interscience, February 1998.

[11] Saul Hansell. Internet is losing ground in battle against spam. *The New York Times: Technology section, 22 April 2003*, 2003.

[12] Evan Harris. The next step in the spam control war: Greylisting. Available at `http://projects.puremagic.com/greylisting/`, July 2003.

[13] Ironport Systems, Inc. Products and services: Messaging gateway appliances. Available at `http://www.ironport.com/products/`, 2003.

[14] John Levine, Ray Everett-Church, and Greg Stebben. *Internet Privacy for Dummies*. Wiley Publishing, Inc., August 2002. Paperback edition.

[15] John Leyden. Spam epidemic gets worse. *The Register*, 55(34331), December 2003. Available at `http://www.theregister.co.uk/content/55/34331.html`.

[16] Mail Abuse Prevention System LLC. Getting off the MAPS RBL, 2003. `http://mail-abuse.org/rbl/getoff.html`.

[17] MessageLabs. MessageLabs Monthly View, November 2003. Published on the MessageLabs site, `http://www.messagelabs.com`, November 2003.

[18] MessageLabs. Spam and viruses hit all time highs in 2003. Published on the MessageLabs site, `http://www.messagelabs.com`, December 2003.

[19] Glen Nakamura. The GNU C Library – Absolute Priority. Available at `http://www.imodulo.com/gnu/glibc/Absolute-Priority.html`, 2000–2003.

[20] Martin O'Neal. Corsaire security advisory: Clearswift MAILsweeper MIME attachment evasion issue, March 2003.

[21] Jonathan B. Postel. RFC 821: Simple Mail Transfer Protocol. Available at `http://www.ietf.org/rfc/rfc821.txt`, August 1982.

[22] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A Bayesian approach to fi ltering junk email. Technical Report WS-98-05, AAAI, July 1998. AAAI Workshop on Learning for Text Categorization, Madison, Wisconsin.

[23] Sendmail Consortium. Welcome to sendmail.org. Freeware version of Sendmail available from `http://www.sendmail.org`, 2001–2003.

[24] Sendmail Inc. Sendmail site. Commercial version of Sendmail available from `http://www.sendmail.com`, 1999–2003.

[25] Matt Sergeant. Internet level spam detection and SpamAssassin 2.50. In *Proceedings of the 2003 Spam Conference, Cambridge MA*, January 2003. SpamAssassin is available for download from `http://www.spamassassin.org/index.html`.

[26] SOPHOS Plc. SOPHOS anti-virus and anti-spam for business. Home page: `http://www.sophos.com/`, 2003.

[27] Spam-Blockers.com. Email blacklist directory. Available at `http://www.spam-blockers.com/SPAM-blacklists.htm`, 2003.

[28] Spamhaus Project. Virus and dDoS attacks on Spamhaus. Available at `http://www.spamhaus.org/cyberattacks/index.html`, December 2003.

[29] Sender policy framework, 2004. `http://spf.pobox.com/`.

[30] Symantec. Symantec Security Response: W32.Mimail.F@mm. Available at `http://www.symantec.com/avcenter/venc/data/w32.mimail.f@mm.html`, November 2003.

[31] Symantec. Symantec Security Response: W32.Sobig.F@mm. Available at `http://www.symantec.com/avcenter/venc/data/w32.sobig.f@mm.html`, August 2003.

[32] TarProxy Project. Tarproxy version 0.29 (dev). Available at `http://sourceforge.net/projects/tarproxy`, June 2003.

[33] MailScanner team. Mailscanner version 4.24-5. Available at `http://www.mailscanner.info`, October 2003.

[34] The Turtle Partnership. World data and buzzwords: Messaging. Available at `http://www.turtleweb.com/turtleweb.nsf/list1lookup/MarketData`, 2001–2003.

[35] Wietse Venema. The Postfi x home page. Available at `http://www.postfix.org/`, 1998–2003.

[36] Matthew M Williamson. Design, implementation and test of an email virus throttle. In *Proceedings of ACSAC Security Conference*, Las Vegas, Nevada, December 2003. Available from `http://www.hpl.hp.com/techreports/2003/HPL-2003-118.html`.

[37] Paul Wood. The convergence of viruses and spam: lessons learned from the SoBig.F experience. Available at `http://www.security.iia.net.au/downloads/sobigwhitepaper.pdf`, 2003.

[38] Dale Woolridge, James Law, and Moto Kawasaki. The qmail spam throttle mechanism. Available at `http://spamthrottle.qmail.ca/man/qmail-spamthrottle.5.html`, 2002–2003.

[39] Charles Ying. Sendmail::Milter. Available from `http://sendmail-milter.sourceforge.net/`, 2003.