Design, Implementation and Test of an Email Virus Throttle

Matthew M. Williamson
Information Infrastructure Laboratory
HP Laboratories Bristol

HPL-2003-118
June 30th , 2003*

E-mail: matthew.williamson@hp.com

virus throttle,
email virus

This paper presents an approach to preventing the damage caused by viruses that travel via email. The approach prevents an infected machine spreading the virus further. This directly addresses the two ways that viruses cause damage: less machines spreading the virus will reduce the number of machines infected and reduce the traffic generated by the virus. The approach relies on the observation that normal emailing behaviour is quite different from the behaviour of a spreading virus, with the virus sending messages at a much higher rate, to different addresses. To limit propagation a rate-limiter or virus throttle is described that does not affect normal traffic, but quickly slows and stops viral traffic. The paper includes an analysis of normal emailing behaviour, and details of the throttle design. In addition an implementation is described and tested with real viruses, showing that the approach is practical.

* Internal Accession Date Only

Approved for External Publication

Design, Implementation and Test of an Email Virus Throttle

Matthew M. Williamson
HP Labs Bristol, Filton Road, Stoke Gifford, BS34 8QZ, UK
matthew.williamson@hp.com

Abstract

This paper presents an approach to preventing the damage caused by viruses that travel via email. The approach prevents an infected machine spreading the virus further. This directly addresses the two ways that viruses cause damage: less machines spreading the virus will reduce the number of machines infected and reduce the traffic generated by the virus.

The approach relies on the observation that normal emailing behaviour is quite different from the behaviour of a spreading virus, with the virus sending messages at a much higher rate, to different addresses. To limit propagation a rate-limiter or virus throttle is described that does not affect normal traffic, but quickly slows and stops viral traffic. The paper includes an analysis of normal emailing behaviour, and details of the throttle design. In addition an implementation is described and tested with real viruses, showing that the approach is practical.

1 Introduction

Computer viruses and worms[1] are a continual headache for all computer users. They cause damage to users' machines, tie up infrastructure and resources, and waste the time of IT staff. Most approaches to viruses rely on detecting the "signature" of the virus, and as such need to be continually updated as new viruses are released. While the signature is being generated the virus is able to spread unhindered, and if that delay is long, the virus can cause significant damage.

This paper extends previous work on virus throttling (Williamson, 2002) to email viruses. Virus throttling is an approach that limits the damage caused by viruses by targeting the virus' propagation. The technique essentially prevents a machine infected with a virus from spreading that virus further. Targeting propagation is a sensible approach because it directly addresses the two ways that viruses cause damage: fewer machines will become infected because fewer machines will be spreading the virus, and the load on network infrastructure will be lower as the virus will generate less traffic. In addition, by concentrating on the virus behaviour as opposed to its exact form, a virus throttle works on previously unknown viruses and needs no signature updates.

Virus throttling is based on the observation that the traffic generated by a spreading virus is quite different from normal traffic. A virus will send messages to many different destinations as quickly as possible, but normal traffic is more leisurely and there is more locality—messages are sent repeatedly to the same destinations. The throttle enforces a limit on the rate that messages can be sent to different destinations, so that normal traffic is passed but traffic from a virus is slowed and stopped. The rate-limit is "soft" in the sense that high rate messages are delayed not dropped in order to enforce the limit. This means that false positives (when the throttle would limit legitimate traffic) result in small delays to messages not a loss of service.

Of course there are occasions when normal email traffic will look like a virus, e.g. many messages to different destinations in a short period of time. An inadvertent reply-to-all is a good example of this. In those cases the user would be contacted to say that the throttle was delaying the mail, and could override the throttle. In addition there are some applications e.g. bulk mailers that could not be throttled. This just means that should those machines become infected with a virus, the virus could not be hindered as well as on other machines.

The throttle is intended to rate-limit outgoing email messages from client machines, i.e. as the mail enters the email routing system, so is best implemented at an outgoing mail server. In that case the throttle would be effective against machines sending spam, as spam sent through the server would be indistinguishable from a virus: they both consist of many messages to different recipients at a high rate.

Other approaches to email viruses concentrate on trying to deduce if incoming email messages contain viruses, so protecting the machine from infection. This is commonly achieved using signatures e.g. products from Symantec (2003), or using heuristic rules (Shipp, 2002), or by running the email in a sandbox (also known as behaviour blocking), Messmer (2002); Okena (2002). The throttle differs from these approaches because it concentrates on outgoing rather than incoming messages, attempting to prevent the virus spreading further rather than preventing infection. A related product is the HAWK (Hostile Activity Watch Kernel)

[1] In this paper the terms are used interchangeably.

from McAfee (2003), which prevents a virus sending mail to a large number of addresses in a short period of time.

For spam there are a variety of approaches that attempt to rate-limit messages, either from servers on a blacklist (Teergrube, 2003), or by using a tarpit (Cleaver, 2003).

The rest of the paper describes the characteristics of email traffic that make it suitable for throttling and the design of the throttle. The effect of the throttle on normal traffic is then presented, showing that the effect is small. The implementation of an SMTP (see Postel (1982)) based throttle is then described together with results of testing with real email viruses. The paper concludes with predictions of the server performance and more general conclusions.

2 Characteristics of Email Traffic

An email virus throttle is a rate-limiter on outgoing emails to different destination addresses. The claim of this paper is that normal traffic tends to be at a low rate so would be largely unaffected by the rate-limiter, but a virus that sends email at a high rate to different addresses would be limited and stopped. The trick is to design the filter or throttle so that is has minimal impact on normal traffic, but maximal impact on viral traffic.

Previous work used a throttle which consists of two independent processes (Williamson, 2002). One processing loop checks outgoing messages to see if they are at lower rate than allowed, and if so they are passed as normal. If not, they are placed on a “delay queue”. A second process pops messages off the delay queue at a fixed rate, so enforcing the rate limit. If a virus infects the machine, it will attempt to send messages at a high rate and the queue will grow large. This is easily detected and the onward propagation can be stopped. The reason for using the queue and delays is to tolerate errors in the detection mechanism—occasional bursts of high rate traffic will be delayed slightly. In addition using the delay allows the virus to be rate-limited even before it is stopped. This is particularly important for email viruses as every email sent is likely to go to a valid address (compared to IP based viruses such as Nimda (CERT, 2001) where a random IP address is not likely to resolve to a machine, and is thus less likely to be harmful).

The exact design of the throttle depends on the characteristics of normal traffic. To that end data was collected from the “Sent Items” folders of 22 users, making sure that that folder was an accurate record of that user’s emails activity. The users’ roles spanned engineers, admins, managers and IT staff. In all the data consisted of 33084 emails to 61749 recipients over a cumulative period of 18.2 years.

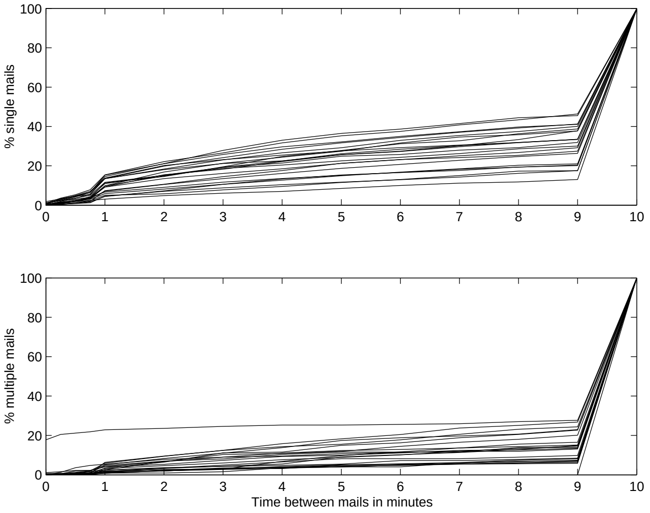


Figure 1: Time differences between subsequent mails. The x-axis is the time between emails in minutes, and the y-axis the percentage of emails that occur within that time period. The different lines correspond to different users. The top plot is for single recipient mails, and the lower for multiple. Single recipient mails tend to be sent closer together than multiple mails, but overall a small proportion of mails is sent quickly (>1 per minute). The outlier in the lower graph is a member of IT staff who occasionally sends messages using an automated script. The plot shows that this behaviour is quite different from normal (human) emailing activity.

Since the throttle is a rate-limiter, the first issue is the rate that emails are sent. Email traffic is bursty with long periods of inactivity (e.g. during the night), so rather than calculating a frequency, Figure 1 shows a plot of the time differences between subsequent mails. The top plot shows times for mails to single recipients, and the lower plot for mails to multiple recipients. These are treated differently because a multiple recipient mail is effectively the same mail to different recipients, and so could look like a virus. While present day viruses do not send mail to multiple recipients, there is no reason why they might not in the future.

The figure shows that there is considerable variation between users, but overall the percentage of mails that are sent at a high rate (>1 per minute) is low. Multiple mails tend to be sent at a lower rate, e.g. 5–15% of multiple mails are sent within 5 minutes as opposed to 10–40% of single recipient mails

In order for a virus to spread, it needs to send itself to different addresses (it makes no sense to re-infect the same machine), but in normal mailing users mail the same address repeatedly. This locality was exploited in previous work (Williamson, 2002) to only rate-limit messages to destinations different from those contacted recently.

Figure 2 shows a measure of locality for the email data. The plot shows the percentage of emails sent to an address which is the same as one in the previous n addresses mailed, for different values of n. For example if $n = 1$, this is the proportion of occasions that two mails we sent consecutively to the same address. The plot shows that single recipient mails have higher locality than multiple mails, but neither has the kind of locality evident in other types of traffic (e.g. http traffic described in (Williamson, 2002) has approximately 90% of traffic to addresses in the previous 5).

While this data suggests that locality is not important, it

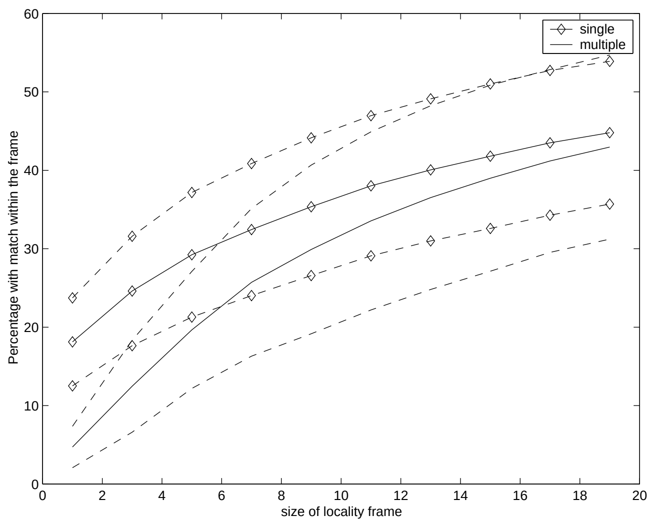


Figure 2: Percentage of emails with addresses the same as addresses emailed recently. The x-axis is the number of previous addresses, and the y-axis the percentage of emails with a match. The lines represent the mean and one standard deviation above and below for single recipient mails (diamonds) and multiple mails (lines). Single recipient mails are more likely to be addressed to an address mailed recently, but in neither is the effect particularly pronounced.

still makes sense to rate-limit messages to different addresses. This is because such a filter is more likely to catch a slow spreading virus, and in addition would allow repeated "conversational" emails (where email is used more like instant messaging) to occur without delay since that behaviour consists of high rate messages to the same destination.

This data was collected in a corporate environment, but one would expect to find similar characteristics of "consumer" mail, i.e. mails sent at a low rate (they take time to compose), with some locality. One difference might be if it were common for messages to be composed offline, and users connect to send them. This would result in small burst of messages, which might look like a virus. However if each burst is small, the overall effect will not be large.

3 Email throttling algorithm

Given the characteristics of email traffic, the throttling algorithm is shown in Figures 3 and 4.

All outgoing messages from a particular user are throttled. If the message is to a single recipient, it is compared to a short list of recently mailed addresses (the working set). If the address is in the set, it is sent immediately. The working set thus implements the locality effect as described above. If the address is not in the set, then the value of the parameter "slack" is checked.

The slack is a measure of the number of time periods over which no mail has been limited. The user gets credit for not having mail being limited, and can use that credit to send mails. For example, if no mail has been limited for 10 minutes, with an allowed rate of 1 mail per minute, the first mail (or more generally the first maxSlack mails) is allowed through without delay.

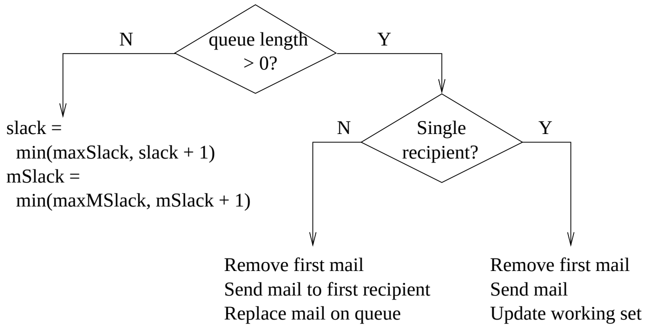


Figure 4: Processing of the delay queue. At regular intervals mails are removed from the queue and sent. If the queue is empty, the slack or "credit" is incremented up to its maximum.

If there is some "slack", i.e. slack > 0, then the mail is sent immediately and the slack decremented. The working set is updated by adding the new address and removing an old one. A variety of replacement strategies can be used (e.g. first in first out, or least recently used). If there is no "slack" then the mail is placed on a queue (the delay queue) waiting to be processed.

The slack essentially guarantees that the average rate of mails will be no higher than the allowed rate. For multiple mails, where the locality effect is less than for single recipient mails, the locality effect is ignored, and a separate slack parameter is used to create a rate limit on the number of *recipients* per minute (see Figure 3). This slack is calculated as before, for example if no mail has been limited for 10 minutes, then a multiple mail with less than 10 recipients will be sent immediately. A mail to more recipients will be split, the message will go off to the first 10 immediately, and the remainder will be queued on the delay queue.

Figure 4 shows the processing loop for the delay queue, which at regular intervals removes the message at the head of the queue, sends it, and updates the working set if it was a single recipient mail. The figure also shows the calculation of the slack parameters—if the queue is empty then the slacks are incremented up their respective maximums.

As mentioned previously, if the delay queue gets large it is evidence that a virus is attempting to spread, but might also be due to legitimate activity, for example a message with an unusually large number of recipients. A threshold on the delay queue length is a reasonable means of detecting such an event. Every time a message is added to the queue, its length is checked (see Figure 3). If the length is greater than the threshold the throttle will stop all further mails being sent (by not accepting any new mails, and not processing the delay queue), and contact the user. The user can then use some other means (e.g. a web interface) to check the mails held at the throttle and either delete them or release them to be sent immediately.

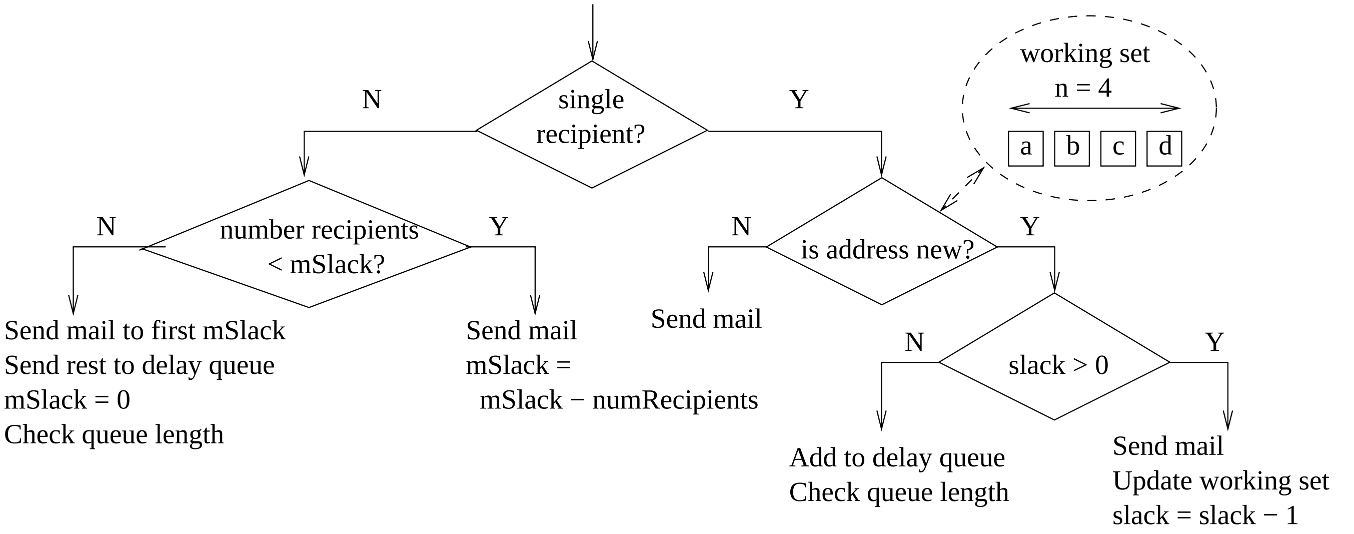


Figure 3: Processing loop for new requests. Each email message is checked through this process and either immediately sent or placed on the delay queue to be processed later.

4 Throttle performance on normal traffic

The parameters of the throttle are the working set size, allowed rate, maximum values for the slack and mSlack, and the value of the stop threshold. This section gives results of the effect of each of these parameters on normal email usage. To best combat viruses all these parameters should be as low as possible.

Figure 5 shows the delays for single recipient mails as a function of working set size, with an allowed rate of 1 mail per minute, and a maximum value of slack of 1. The delays are shown with three different measures, the percentage of mails that are delayed, the average delay per mail sent, and the average delay incurred by mails that are delayed. The graphs show that even with no working set at all the number of delayed mails is low, and the delays themselves are low. There is considerable variation between users. The effect of the working set is not particularly strong (not surprising given the locality results in Figure 2), but it does have some effect. A reasonable value for working set size would be about 5.

Figure 6 shows the effect of the maxMSlack parameter on the delays for multiple mails, with the same measures as before. The throttle is less effective at passing multiple mails without delay, for example with maxMSlack = 15, most users have under 5% of their mails delayed, but some have 20%. However, the average delay per mail is generally low (well under a minute for most users), and the actual delays are again not large (mostly $<$ 10 minutes). This does mean that any email to more than 15 recipients it guaranteed to be delayed slightly, so this parameter could be varied on a per user basis. It is however important to keep this parameter low as it is the number of “free” recipients an email virus could attempt to infect.

Figure 7 shows the effect of the allowed rate on delays for both single and multiple messages combined. For ease of im-

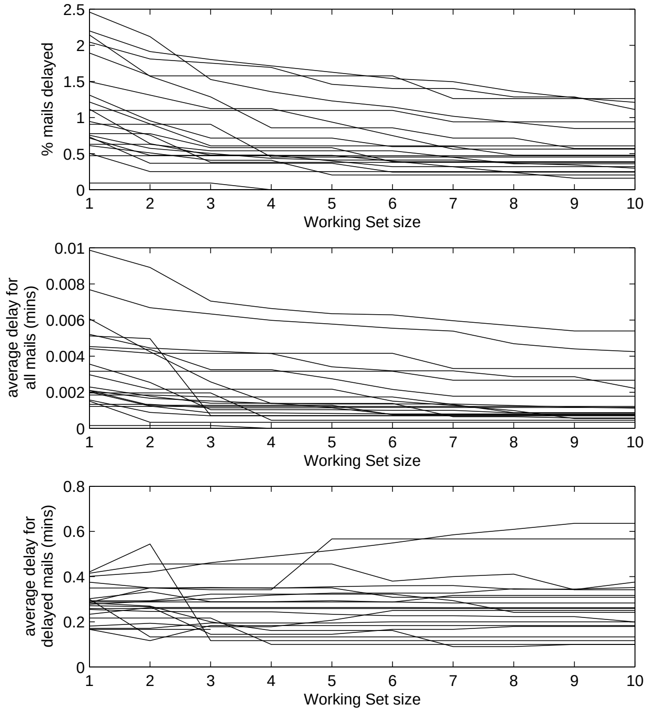


Figure 5: Delays for single recipient mails as the working set size is varied. The plots show three different measures of delay: the percentage of mails delayed (top), the average delay per mail sent (middle) and the average delay of those mails delayed (bottom). The different lines correspond to different users. The plots show that for some users the effect of the working set is strong, but for others it makes little difference. Overall the delays are small, with delays on mails in the range of 10–25 seconds. For this plot maxSlack = 1 and the allowed rate was 1 per minute.

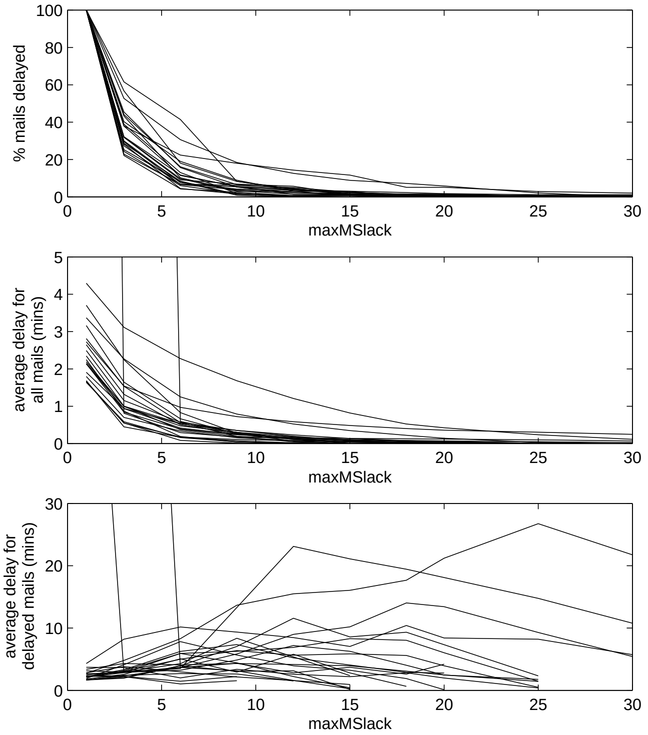


Figure 6: Delays for multiple mails as a function of maxMSlack. This parameter is the maximum amount of “credit” that a user can build up in order to allow a multiple recipient mail to be sent. It is also the maximum size of multiple mail that can be sent without delay. As this parameter increases the delays go down, with the value of 15 giving reasonable delays for all users. The worst affected user is an administrative assistant. The bottom plot is noisy for higher values of maxMSlack because so few mails are delayed. For this plot the allowed rate was 1 recipient per minute.

plementation the allowed rates are made the same for both types of email (although there is no reason in principle why they could not be different). The plot shows that a value of approximately 1 mail/minute or 1 recipient/minute is reasonable, giving delays of under 5 minutes for most users, and up to 10 minutes for some, with around 0.5–3% of mails delayed. Reducing the allowed rate to say 1 mail every 2 minutes increases the number of mails delayed significantly.

The final parameter is the stop limit. Figure 8 shows the number of false alarms (whether the throttle stops legitimate mailing activity) per month that each user would experience as a function of the threshold value. There is significant variation between users, so for example a threshold of 20 would give no false alarms for most users, and occasional alarms for some. One user would have one alarm per month, which is again not excessive.

For reasonable settings of these parameters, the delays for single and multiple mails are generally low (< 5 minutes for most users), and only occur for a small proportion of emails. The question is whether these delays are acceptable. They are certainly of the same order as the transit time for emails

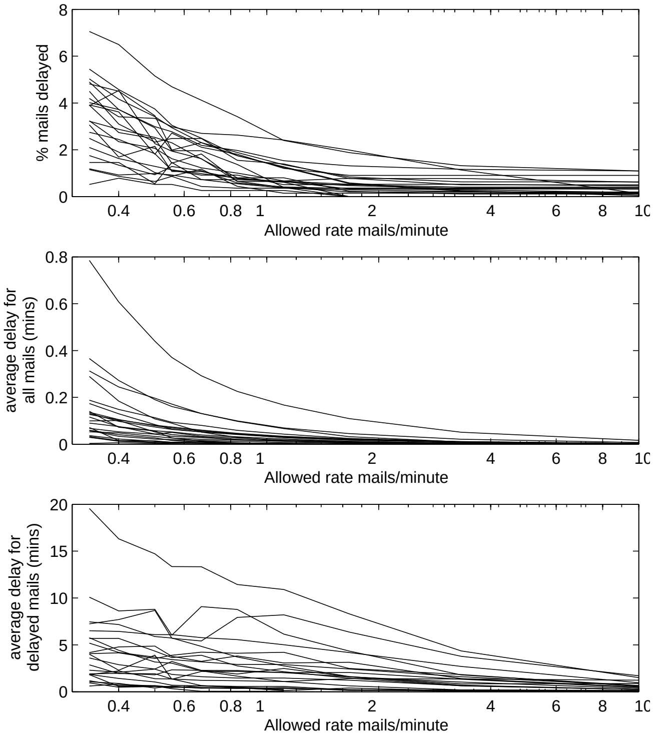


Figure 7: Delays for allowed rate, for both single and multiple mails combined. The rate is shown on a logarithmic scale. The lower the allowed rate, the worse the delays and vice versa.

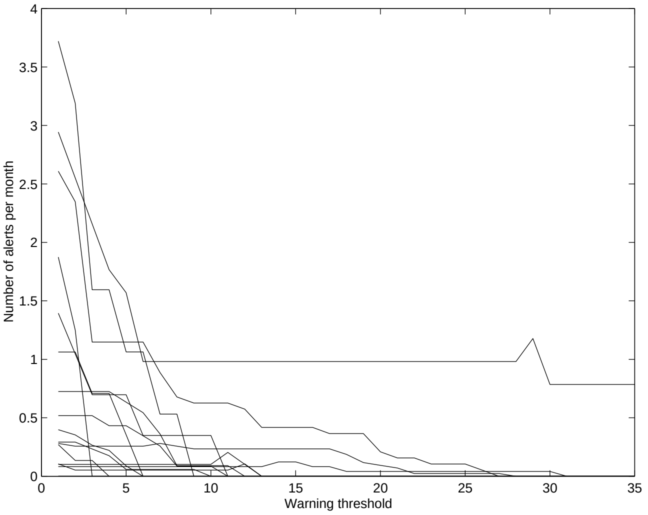


Figure 8: Effect of stop threshold. The plot shows the number of false alarms per month that different users would experience for different values of the threshold. For most users a threshold of 15–20 would give no alarms, while for others the rate would be around 1 every 3–6 months. One user (an administrative assistant) would have false alarms every month, due to sending an email to a large number of recipients. While it is easiest if there is a single set of parameters for all users, it is possible to customise parameters on a per-user basis.

through the email system, and would to a large extent be invisible to the sender (they occur after the mail has been sent, and the sender does not normally know when the email will be read by the recipient). A recent study of timings in email (Tyler and Tang, 2003) showed that the speed that users replied to mails was highly context dependent, e.g. they reply quickly to senior managers, but slower to more lowly colleagues. Given that the delays from the throttle are not large, and users can always override it if necessary, the delays introduced by the throttle should not have a large effect on the usability of the email system.

5 Implementation

In order to throttle email effectively, the throttle needs to be able to determine the sender of each email so that it can enforce the allowed rate per sender. If a virus could "spoof" the sender then it could send messages at a higher rate than allowed by making them appear to come from different users.

Reliably identifying the sender informs both where the throttle should be implemented (best at the point where email enters the email system as after even one hop reliable sender information is lost), and how the sender is identified.

Figure 9 shows a schematic of outgoing mail systems. Microsoft Outlook clients use a proprietary protocol to send mail to an Exchange server (Microsoft, 2003), which then forwards mail for remote delivery. To throttle this protocol the throttle could either be installed on the client or inside the exchange server. While installing the throttle on the client is appealing because it would prevent any network traffic from the virus, it would only guard against mail sent through the client itself. While many viruses have exploited weaknesses in Outlook (e.g. LoveLetter (CERT, 2002)), increasingly viruses are using their own SMTP engines e.g. Klez and Yaha (Sophos, 2003a,c).

The non-Microsoft world uses SMTP, in which case the throttle is best placed at the server. The SMTP routing infrastructure is flexible, allowing intermediate email relays to be inserted in the flow of mail. The throttle can be easily implemented as an extra server that sits between the mail client and its normal outgoing mail server. This server would then implement a copy of the throttle for each user sending mail through that server.

As mentioned above, increasingly viruses are providing their own SMTP engines, and not using the mailer on the client. This begs the question of how they find SMTP servers to send the mail. They have three strategies at present, to use the servers configured for any existing mail clients, to use "known" servers on the Internet pre-configured in the virus, or to attempt to send mail to `smtp.domain` for each `domain` of each email address found on the users machine. This actually suggests that an IP based throttle (Williamson, 2002) could be quite effective at preventing this behaviour!

The second issue is how to identify the user. Unfortunately the sender's address can be easily spoofed (Klez (Sophos, 2003a) was one of the first viruses to do this) and so cannot be relied on to verify the origin of the email. A second candidate is the IP address of the sender's machine. While IP addresses are often dynamically assigned and do change, the throttle does not need much state (working set and slack parameters) and so could work effectively on transient addresses. However there is the difficulty of contacting the user[2]. Other than using the sender's address, which may or not be correct, the alternative is an extra protocol and software on the client to inform them that the throttle has stopped their mail.

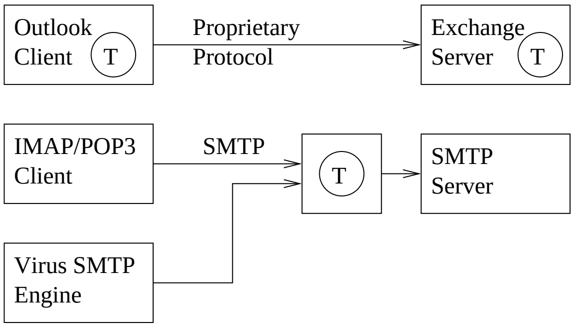


Figure 9: Schematic of an email system for outgoing mail, with suitable positions for an email throttle marked with a "T". With Microsoft Exchange, the best position for the throttle is at the server, although it would also be possible to implement it at the client. For SMTP the logical place for a throttle is at the server, and it could easily be implemented as an extra mail relay. This has the advantage that email viruses with their own SMTP engines would be throttled.

The best solution is to use another mechanism to guarantee that the sender of the mail is registered with the mail server. One such mechanism is authenticated SMTP (SMTP AUTH (Myers, 1999)), where the client provides a username/password pair when sending mail. While the protocol is not particularly secure (e.g. the password is sent unencrypted) the threat being addressed is a virus sending mail masquerading as another user in order to evade the throttle. Sniffing the passwords of other users is possible but would be difficult to automate reliably in a virus.

The advantage of SMTP AUTH is that users of the server need to register with it, and so can provide contact details etc.. The disadvantage is the extra complexity required both at the client (one more configuration step) and at the server (usernames and passwords to manage).

An email throttle was implemented as an SMTP relay using SMTP AUTH, building on top of the JAMES (Java Mail Server) project (JAMES, 2003). This is a production quality open source mail server. The architecture of the modified throttle server is shown in Figure 10.

[2] It is also possible that a virus could use unused IP addresses to masquerade as many machines. This is more complicated than just spoofing the sender's IP. SMTP works over TCP, so the virus would have to really "adopt" the false IP addresses in order to receive the TCP acks, and be able to send messages.

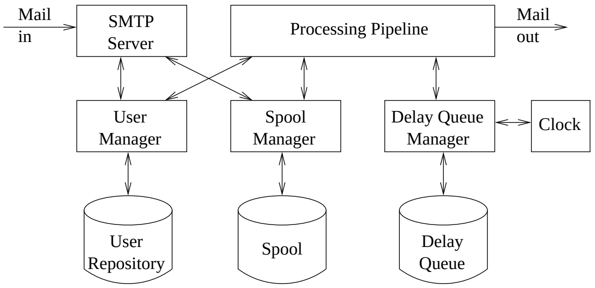


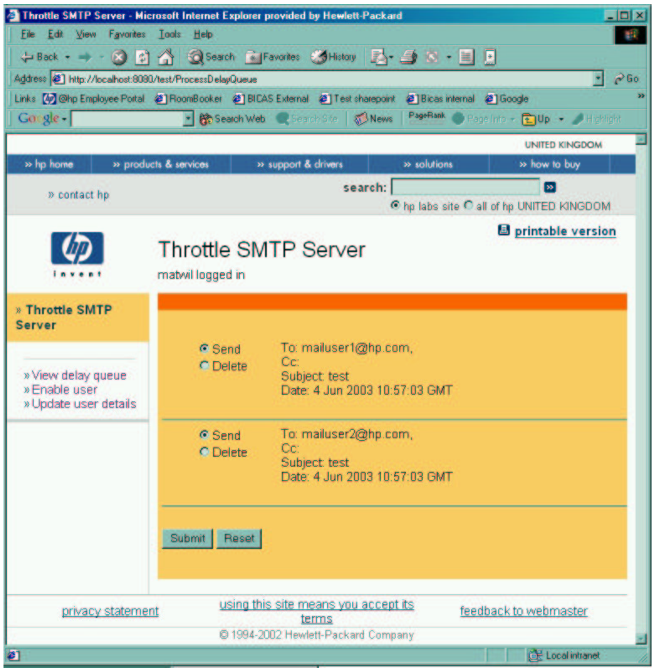
Figure 10: Figure showing the architecture of the email throttle. Incoming mail is placed in the Spool, and then processed by the pipeline, which implements the throttle logic. A DelayQueueManager object is responsible for all requests to the queue itself (simply a store of emails), as well as processing the queue regularly.

Incoming mail is handled by an SMTP server that uses details of users stored in the UsersRepository to to authenticate the sender, and places incoming mail in the Spool. A flexible processing pipeline is then used to process the mail. This consists of a variety of components to select mail messages and perform operations on them. The throttle logic of Figure 3 is implemented in this pipeline. The delay queue is implemented in similar way to the Spool as a store of emails, with a DelayQueueManager object responsible for all interactions with the delay queue as well as processing the queue (Figure 4) regularly. The user state (working set, slack, mSlack, etc.) is stored in the UsersRepository. The program is easily configurable using an XML configuration file that allows for example the setting of parameters, the configuration of the pipeline, and the choice of using the file system or a relational database to store the emails and user information.

The servers uses the email address in the message and so does not differentiate between addresses that map to single recipients and those that map to mailing lists that may be expanded downstream. Expanding mailing lists would make the throttle better, but would add significant extra complexity.

If the delay queue length is larger than the stop threshold for any particular user, the server refuses any new mails from that user (the authentication fails). In addition the delay queue processing is suspended for that user. To contact them, the throttle server sends an email to an address configured when the user registers with the server, the message containing details of the mails waiting in the mail queue. To delete or send these messages the user can either send an especially crafted email to the server, or can visit a web interface (screen-shot in Figure 11). The web interface can be used for other administration tasks, e.g. changing passwords, user details, etc..

The implementation is currently being beta tested in a small scale trial, and was used to test the performance of the throttling algorithm with real viruses, as described in the following section.



http://localhost:8080/test/ProcessDelayQueue

Throttle SMTP Server

matwil logged in

● Send ○ Delete
To: mailuser1@hp.com,
Cc:
Subject: test
Date: 4 Jun 2003 10:57:03 GMT

● Send ○ Delete
To: mailuser2@hp.com,
Cc:
Subject: test
Date: 4 Jun 2003 10:57:03 GMT

© 1994-2002 Hewlett-Packard Company

Figure 11: Screen-shot of the user interface for the virus throttle. The interface allows the user to see the contents of their delay queue, and specify which mails will be deleted and which sent immediately.

6 Tests on real viruses

In order to test the efficacy of the throttle on real viruses, the isolated testbed shown in Figure 12 was constructed. It consists of four Linux machines. The first (`host.virus.net`) runs VMware (VMware, 2003) with a vulnerable Windows 2000 image. The outgoing mail server on the Windows 2000 machine is set to be the throttle box (`smtp.virus.net`), and the address book and inbox were populated with email addresses. The throttle server is configured to pass all mail on to another SMTP server (`blackhole.virus.net`) that simply saves mail to disk. To deal with the fact that some viruses find the mail server by prepending “smtp” to email domains found on the infected machine, a DNS server was set up on the last machine to provide the `virus.net` domain.

The test consisted of launching the virus on the Windows machine and detecting how quickly the throttle could stop the virus, as well as how many emails were forwarded. None of the viruses tested used SMTP AUTH, so the throttle was altered to use the IP address of the sending machine as the origin of the mail.

Table 1 shows the results for two real viruses Yaha.E (Sophos, 2003c) that sends mail to addresses found in the address book, and Lovgate.A (Sophos, 2003b) that uses the inbox. Also included are results from a test virus (a program that sends emails to different addresses at a pre-determined rate). The parameters of the throttle were maxSlack 1, maxMSlack 15, working set size 4 and the stop limit was 20.

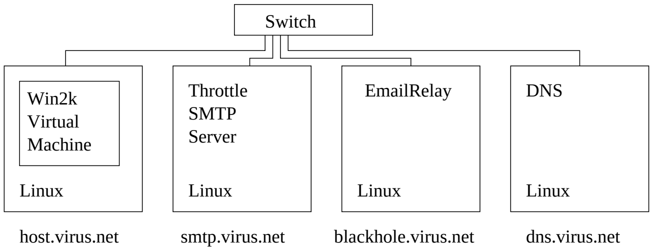


Figure 12: Figure showing the testbed for virus tests. This consists of four Linux machines, one running a machine vulnerable to viruses (`host.virus.net`), the throttle server (`smtp.virus.net`), another email relay that captures any forwarded mail from the throttle (`blackhole.virus.net`), and a DNS server (`dns.virus.net`).

Table 1: Stopping times for real viruses. The table gives the number of passed mails and the stopping time for two real viruses and a test virus. The results are the averages of 3 runs of each real virus, and a single run of the test virus. The throttle stops viruses very quickly, and even stops quite slow spreading viruses fairly quickly (20 minutes for one spreading at 2 emails per minute).

Virus	Approx	Number	Stopping time
	rate/min	passed	(min:sec)
Lovgate.A	109	∼ 3	0:12
Yaha.E	455	∼ 0	0:03
Test	60	∼ 2	0:24
Test	10	∼ 6	2:33
Test	5	∼ 6	5:01
Test	2	∼ 21	20:04

The table shows that the throttle is very effective at stopping real viruses, stopping them in under 15 seconds, with few emails forwarded. Yaha.E sends no messages because it appears to need the user to send a message to “start” it. That first message uses up any “slack”, so that all the messages from the virus go straight onto the delay queue. This quickly reaches the threshold and the virus is stopped.

The table also shows that even if the virus attempts to spread slowly (e.g. 2 emails/minute) the throttle will eventually stop it, and still fairly quickly. The delay queue mechanism effectively integrates the difference between the virus spreading rate and the allowed rate, so that any virus with a spreading rate that is slightly above the allowed rate will eventually be stopped.

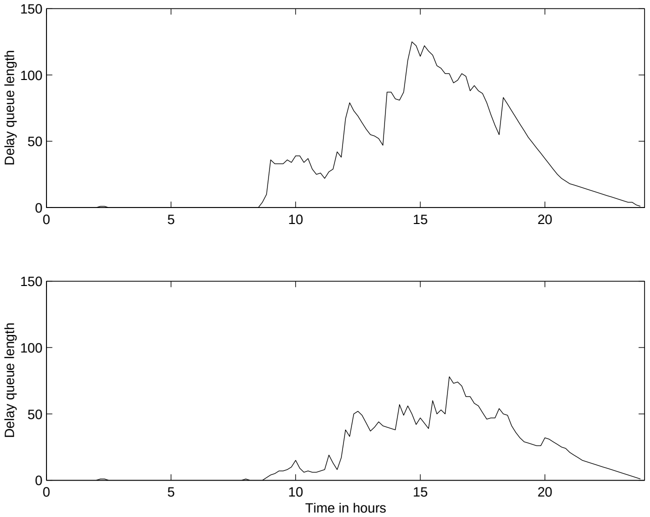


Figure 13: Cumulative delay queue lengths for a throttle server with 500 users. The plots give two examples of typical activity. The delay queue does not grow large, in spite of the large numbers of users.

7 Server Performance

An SMTP server implementing a virus throttle has to perform more work than an ordinary SMTP server: it needs to read and update state for each user, as well as store emails in the delay queue.

Reading and writing state occurs for each processed mail and required interactions with the user database. While no firm performance figures are currently available, during the virus experiments described in the previous section were being run, the server did not appear to be overloaded handling 450 mails per second[3].

SMTP servers often store mails while waiting to deliver them, but it is important that the extra storage required for the delay queue does not cause problems. To test this, the effects of a large number of users using the throttle server was simulated by taking the data originally collected, dividing the time history into 24 hour chunks, and choosing 500 days at random across the users. The throttle was then simulated for these 500 “users” in parallel, so approximating normal usage of a mail server.

Figure 13 shows two typical examples of the length of the delay queue across a 24 hour period. As one might expect the length of the queue varies with the time of day but seldom grows large, because few of the emails are delayed.

This result suggests that the extra loading of an email server implementing a throttle is likely to be minimal.

8 Conclusions

This paper has presented an approach to limiting the damage caused by email worms and viruses. The technique targets the fundamental behaviour of a virus, to propagate by sending messages to many different addresses. The paper has shown that normal email traffic does not have this property, and that a throttle or rate-limiter can be designed that both lets through normal traffic without much delay, and slows and stops viral propagation. A working implementation has been described and tested against real email viruses. The implementation has proved effective at preventing further spread, stopping real

[3] Run on a 1.7GHz Pentium III, Redhat Linux 7.3, with the throttle server and MySQL database (MySQL, 2003)

viruses in well under a minute. The paper has also presented evidence that the throttle is likely to scale to handling a large number of users.

A further benefit of the approach is that it would be effective at limiting spam. If an ISP ran the throttle, spammers would not be able to send mail at high rates.

The technique is an altruistic one—the machine still becomes infected, just does not spread the infection further. A consequence of this is that it needs to be widely deployed in order to have a large effect. There are two immediate effects even with partial deployment: the traffic created by a spreading virus is greatly reduced, so those servers that implement the throttle will not be affected by overloading during an attack; and if the virus is only slowed slightly, it will improve the effectiveness of the overall response (throttling plus signature based methods) significantly (Williamson and Léveillé, 2003).

Were throttling to be widely deployed, it would change how viruses were written: they would not be able to cause damage by spreading quickly. This is good because slow spreading viruses are easier to combat with slow responses such as signature based scanning. A consequence of this may be that payloads become more malicious, in which case the techniques of behaviour blocking (Messmer, 2002; Schmid et al., 2002) would be good mechanisms to prevent damage on the host machine.

The general approach taken here can be thought of in two ways. Firstly it concentrates on limiting the damage that a machine can cause to others, rather than limiting the damage that can be done to it (also suggested by Bruschi and Rosti (2000)). Secondly its effect is to mitigate the effects of the problem quickly and automatically, so buying time for a slower response (Williamson, 2003). There are many problems in security and more broadly in IT where problems occur at machine speeds and do large amounts of damage before a (slower) human response can be mounted. Throttling and other technologies (e.g. Somayaji and Forrest (2000)) that attempt to limit that damage and hold off the attack until a human can respond should increase the resilience of our computing systems to attacks, misconfigurations and other problems.

Acknowledgements

This work would not have been possible without the support of my colleagues at HP Labs, Jonathan Griffin, Andy Norman and Jamie Twycross.

References

Bruschi, D. and Rosti, E. (2000). Disarming offense to facilitate defense. In *Proceedings of the New Security Paradigms Workshop*, Cork, Ireland.

CERT (2001). CERT Advisory CA-2001-26 Nimda Worm. Available at http://www.cert.org/advisories/CA-2001-26.html.

CERT (2002). CERT Advisory CA-2000-04 Love Letter Worm. Available at http://www.cert.org/advisories/CA-2000-04.html.

Cleaver, J. (2003). Jackpot mailswerver. http://jackpot.uk.net/.

JAMES (2003). Java mail server. http://james.apache.org.

McAfee (2003). VirusScan Home Edition 7.0. http://www.mcafee.com/myapps/vs7/default.asp.

Messmer, E. (2002). Behavior blocking repels new viruses. Network World Fusion News. Available from http://www.nwfusion.com/news/2002/0128antivirus.html.

Microsoft (2003). Microsoft exchange server. http://www.microsoft.com/exchange/default.asp.

Myers, J. (1999). SMTP service extension for authentication. http://www.ietf.org/rfc/rfc2554.txt.

MySQL (2003). http://www.mysql.com/.

Okena (2002). Stormwatch. http://www.okena.com/areas/products/products_stormwatch.html.

Postel, J. B. (1982). Simple mail transfer protocol. http://www.ietf.org/rfc/rfc0821.txt.

Schmid, M., Hill, F., and Ghosh, A. K. (2002). Protecting data from malicious software. In *Proceedings of ACSAC Conference*, pages 199–208, Las Vegas, Nevada.

Shipp, A. (2002). Desktop AV and Internet level anti-virus - a comparison. *Information Security Bulletin*, pages 33–38.

Somayaji, A. and Forrest, S. (2000). Automated response using system-call delays. In *Proceedings of the 9th USENIX Security Symposium*, pages 185–197, Denver, CO.

Sophos (2003a). W32/Klez-E. http://www.sophos.com/virusinfo/analyses/w32kleze.html.

Sophos (2003b). W32/Lovegate-A. http://www.sophos.com/virusinfo/analyses/w32lovgatea.html.

Sophos (2003c). W32/Yaha-E. http://www.sophos.com/virusinfo/analyses/w32yahae.html.

Symantec (2003). Symantec corporation. http://www.symantec.com/.

Teergrube (2003). Teergrubing faq. http://www.iks-jena.de/mitarb/lutz/usenet/teergrube.en.html.

Tyler, J. R. and Tang, J. C. (2003). When can I expect an email response: A study of rhythms in email usage. In *Proceedings of the European Conference on Computer-Supported Cooperative Work*, Helsinki, Finland.

VMware (2003). VMware Workstation. http://www.vmware.com/.

Williamson, M. M. (2002). Throttling viruses: Restricting propagation to defeat malicious mobile code. In *Proceedings of ACSAC Security Conference*, pages 61–68, Las Vegas, Nevada. Available from http://www.hpl.hp.com/techreports/2002/HPL-2002-172.html.

Williamson, M. M. (2003). Resilient infrastructure for network security. *Complexity*. Under review. Available from http://www.hpl.hp.com/techreports/2002/HPL-2002-273.html.

Williamson, M. M. and Léveillé, J. (2003). An epidemiological model of virus spread and cleanup. In *Proceeding Virus Bulletin Conference*, Toronto, Canada. Available from http://www.hpl.hp.com/techreports/2003/HPL-2003-39.html.