# Virus Throttling for Instant Messaging

Matthew M. Williamson, Alan Parry, Andrew Byde
HP Labs Bristol, Filton Road, Stoke Gifford, BS34 8QZ, UK
matthew_williamson@alum.mit.edu, alan.parry@hp.com, andrew.byde@hp.com

April 28, 2004

## Abstract

Virus Throttling is a technique to slow the spread of worms and viruses by targeting their propagation. It works by preventing an infected machine infecting many others. This then results in fewer machines becoming infected and less traffic generated by the virus.

The technique has been shown to work well for worms that spread over many TCP/IP protocols as well as email. This paper applies the technique to Instant Messaging.

Worms and viruses that spread over Instant Messaging are a growing but significant threat. They have the potential to spread very rapidly causing widespread damage. While there have been few Instant Messaging viruses in the wild, they are likely to become more of a problem in the future, particularly as many enterprises are adopting Instant Messaging for internal use.

## 1 Introduction

Worms and viruses[1] that spread over instant messaging (IM) are a new and potent threat to enterprises and home computer users. The client programs used to read and send messages are increasingly complicated, and thus are likely to contain bugs (or even features!) that can be exploited by malware. For enterprises, IM is particularly worrying as IM protocols often use tunnelling to traverse firewalls, allowing any malicious payload (denial of service, key logging, damage to machines etc. ) to be suffered inside the corporate network. Finally because each user generally has a list of "buddies" it is easy for the worm or virus to find new users and machines to infect. This means that IM worms have the potential to spread very quickly [8].

Current techniques for dealing with worms and viruses, notably patching machines to remove vulnerabilities, and using signature detection methods, are unsatisfactory for this threat. Patching machines is slow, hard work and often incomplete, in spite of efforts to improve it e.g. Windows Update [12]. Indeed, many IM clients are third party software and patching occurs on an ad hoc basis at best. Signature detection methods operate by definition after the malware has been released, and so are forever playing catchup. With the fast propagation rate of IM worms, signature based mechanisms are likely to be stretched, and would provide little protection against unknown (so called zero-day) attacks.

This paper presents an alternative approach based on limiting the propagation of any IM worm. Virus Throttling [18, 16, 19] is based on the observation that normal traffic on many protocols is quite different from the traffic generated by a worm or virus spreading on the same protocol. This paper will show that in the case of IM, the normal interaction of an IM user with their buddies is quite different from a virus spreading by sending messages to all their buddies. In particular, IM usage consists of many interactions with a slowly varying subset of buddies, while a virus will send messages to many different buddies. Restricting the rate that a user can interact with others will contain the virus but not be noticeable to the user.

The paper begins by providing background information about how common instant messaging systems work, and then analyses in detail the structure of a network of IM users, showing that IM worms could indeed spread quickly. Virus throttling is then introduced and its efficacy demonstrated using data collected from the messaging habits of over 700 users. The implementation of throttling at the messaging server is also described. The final sections return to the IM network and show what

[1] While recognising the differences between viruses and worms, in this paper both terms are used to refer to the same thing: propagating malware.

effect throttling would have on the global spread of a worm.

## 2 Instant Messaging Background

Instant messaging is a method for real time communication over the Internet. Instant messaging clients send text, xml or html messages back and forth to form a conversation. It is generally also possible to send files, join chat rooms and access other services e.g. stock quotes using IM. There are a variety of IM systems including MSN Messenger, AOL Instant Message (AIM), Yahoo Instant Messenger (YIM) and Jabber.

There are two main ways that IM systems are architected: server proxy and server broker. In the server proxy architecture, all messages are passed through the server. If two users A and B want to communicate, A sends a message to the server, which passes it to B and vice versa. This is the most common arrangement.

The alternative is server brokering. Here the server only handles the setup of the communication, and the messages are sent directly between the two users. So if A wants to chat with B, she sends a message to the server. The server then contacts B and tells him that A wants to chat. If B agrees, B's contact information (normally the IP address of his machine and a port number) are forwarded to A, and they exchange messages directly.

This mechanism involves less load on the server, but it is often blocked by firewalls. Many systems use both arrangements. For example, the Yahoo Instant Messages first attempts server brokering and if that fails it reverts to server proxying. Server brokering is the main method used for file transfer.

In both cases, the messaging server has control of the initiation of the communication, and in the proxy case, the server also has control over the whole conversation.

## 3 Instant Messaging Networks

Computer worms spread over networks, the particular network determined by how the worm finds the addresses of machines to attack. For IM worms, this means the connectivity of the buddy lists on each users machine. For example, if A has 10 friends in her buddy list, a worm infecting A's computer and spreading over IM would spread to her 10 friends.

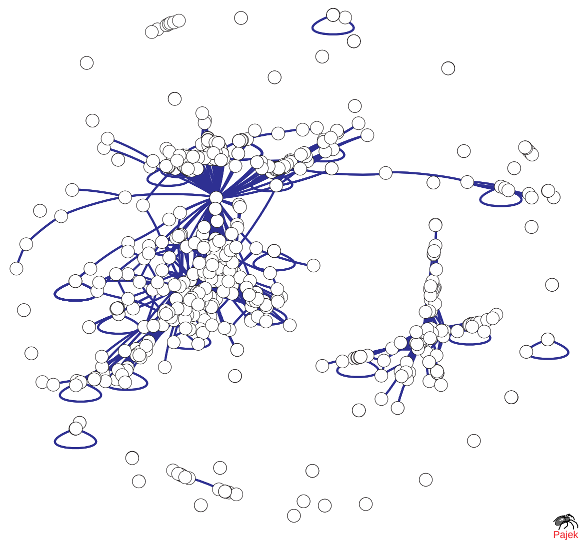


Figure 1: The network of IM users. There is a link between users if they appear in each others buddy lists. Graph plotted with Pajek [3].

We collected the buddy lists of 710 users of a corporate IM server (Jabber [9]) and have analysed them. Figure 1 shows the network structure, showing the connections between the buddies on that server, and ignoring links to buddies hosted on other servers. Rather surprisingly the network forms two large disconnected clusters, and a great number of small clusters where a few users are connected. This means that a single worm cannot infect all the users of this server by spreading within it. These clusters might be connected through the buddy lists of users on other servers, but that would be impossible to deduce from our data. Another interesting aspect is that there appear to be a number of individuals with many connections that lie at the heart of each larger cluster.

There are different types of network, and each type has different properties with respect to how viruses can propagate over them. Examples are random graphs (nodes connected randomly), lattices, small world networks [17] and scale-free networks [14]. Kephart [10, 11] gives a good analysis of how quickly viruses can spread on some of these networks.

Epidemiologists characterise networks with what is called the epidemic threshold. In a fully connected network (each node connected to every other node), if an infected node has a chance $\beta$ of infecting another, and a chance $\delta$ of being cured, then the virus will have a sustained population if $\beta/\delta > 1$. The critical value of $\beta/\delta$ is called the epidemic threshold [13]. With networks that are less fully connected, this threshold may vary, for example $\beta/\delta$ might need to be greater than 0.3 for a virus to establish an epidemic. Scale-free networks are a special class of networks for which the epidemic threshold is zero, i.e. it is very easy for viruses to spread [14].

Because of this property, scale-free networks have been the subject of much study[14]. They have been observed in email contacts [6], the structure of the world wide web [2] and the routing infrastructure [7]. In structure, they are networks where a significant proportion of the nodes are highly connected. More precisely, the proportion of nodes with $k$ links (also called degree $k$) is proportional to $k^{-\alpha}$, where $\alpha$ is a constant. The behaviour of the network is dominated by the highly connected nodes: viruses spread quickly because they can easily reach large parts of the network. The structure also means that any node chosen at random is likely to have a small number of links (the highly connected nodes are a small proportion), so immunising (e.g. with a virus signature) at random has a weak effect on the overall virus spread. Targeting immunisation on the most highly connected nodes has a much stronger effect, indeed only a small proportion of nodes need to be immunised in order to make the network much harder for viruses to traverse [5, 15]. Unfortunately in practise it is quite difficult to find the highly connected nodes, and the exact topology is more determined by how the virus is written: if Code Red [4] had used the URLs in web pages on infected servers to spread, rather than guessing IP addresses randomly, the its topology would have been quite different (something more like scale-free [2], as opposed to fully connected [1]).

One way to determine the type of a network is to plot the degree distribution. This is the histogram of the number of nodes with a certain degree (number of links). A scale-free graph's histogram plotted using logarithmic scales gives a straight line with gradient $-\alpha$.

Figure 2 shows the histogram for the IM buddy list network above. The histogram on the left is plotted with linear axes, and shows that a small proportion of the users have large buddy lists (only 3 users have lists larger than 100). The right hand plot is the same data plotted with logarithmic axes. The linear nature of the plot suggests that this IM network has a scale-free property.

In order to assess the speed that a worm could propagate on this network, a simple simulation was used. At time $t = 0$, a single node was infected. After some infection time $t_{wait}$ the worm will propagate to all the buddies of

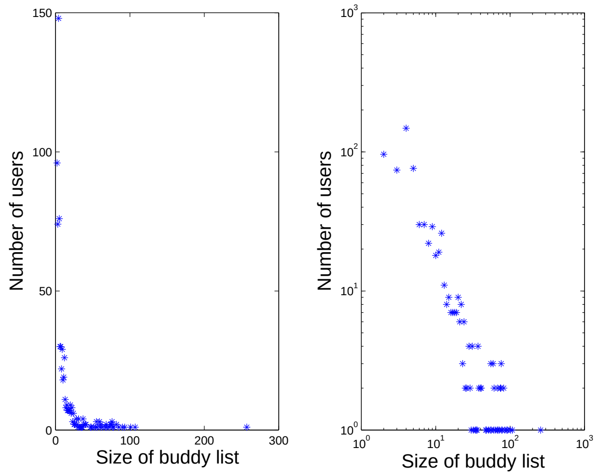


Figure 2: Histogram of number of users with different size buddy lists. The right hand graph is the same data plotted with logarithmic scales.

the infected machine, taking $t_{infect}$ for each neighbour. It is assumed that the infection takes place with a single message. These machines would then wait and infect their buddies in turn. Using conservative estimates of these parameters e.g. $t_{wait} = t_{infect} = 1$ second, and using the real network above, the time course of the infection is as shown in Figure 3.

The plot shows the two large clusters being infected, and also how for some starting infections the overall infection can be very low. It also shows the speed of infection. The time from initial infection to saturation (no more machines infected) is generally less than 20 seconds, with the bulk of machines infected after 10 seconds. The lower plot shows the traffic (in messages/minute) generated by the virus, showing considerable loading on the messaging server during an outbreak.

This is a relatively small network, but one would expect similar characteristics and properties from larger ones. The clusters, and the difference in buddy list sizes reflect the different behaviour of different people and their social interactions. One would thus expect larger IM networks to also exhibit scale free properties, some large disconnected clusters and many small isolated ones. Would a virus spread so quickly in a larger network? It would probably be slower, but not much, because any highly connected nodes allow the virus to quickly spread over large portions of the network.

This analysis of a real IM network thus brings both good and bad news. Firstly the actual outbreak is likely to be smaller than the total number of users (since not ev-

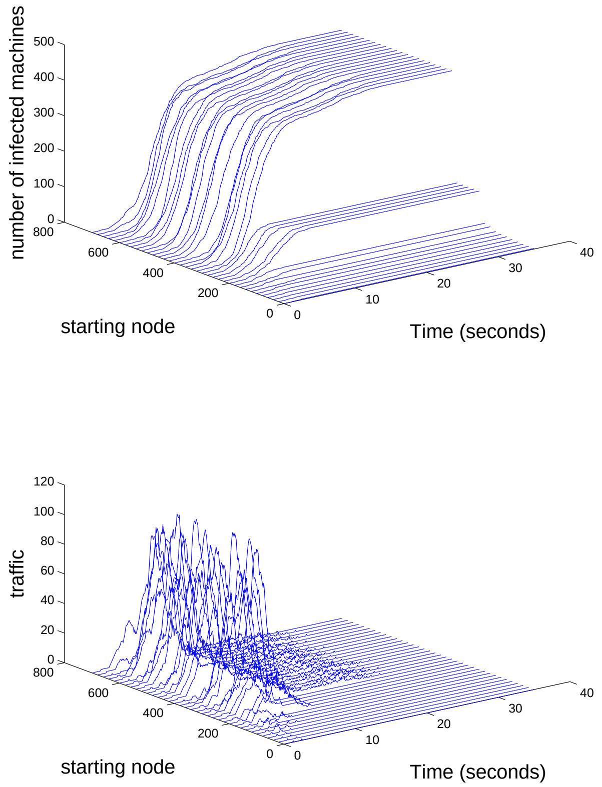


Figure 3: Figures showing the time course of a virus spreading from a sample of nodes in the network. The upper graph shows the number of infected machines, and the lower graph the traffic generated.

ery user is connected to every other one) and might be considerably smaller (there are many small isolated clusters). On the other hand, should an outbreak occur, it is likely to spread very quickly, infecting many machines and causing large loads at the messaging server.

## 4 Virus Throttling

Virus Throttling [18, 16, 19] is an approach that restricts the spread of a worm or virus after it has infected a machine. It is based on the observation that the normal traffic from a machine tends to be directed to a slowly varying set of other machines, and that this is quite different from the behaviour of a virus, which sends messages to many different machines. Restricting the rate that a machine can communicate with "different" machines thus does not impede normal usage, but will restrict the rate that a virus can propagate. Restricting propagation will cause the virus to spread more slowly, giving more time

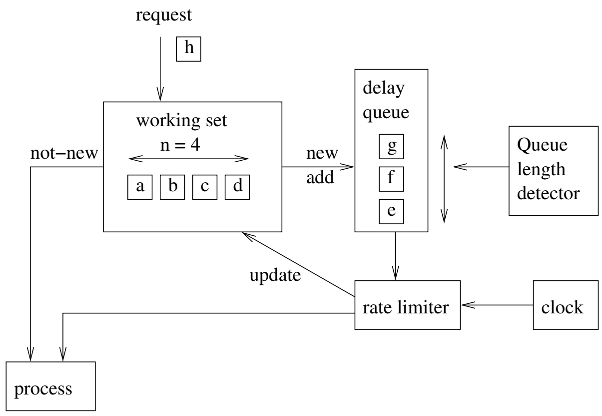


Figure 4: Throttle algorithm. Every time a message is sent, its destination ("h" in the figure) is compared with the working set. If it is in the set, it is passed, if not, it is placed on the delay queue for sending later. Messages are removed from the queue at regular intervals, sent, and the working set updated. If the queue gets too large—evidence that a virus is attempting to spread—all further messages can be blocked.

for slower more definite mechanisms (e.g. signatures), and also reduce the amount of traffic generated by the virus.

Luckily this observation is true for many TCP/IP protocols [18] and email [19]. In order to determine if it also holds for IM traffic, we logged the time, sender and recipient of all messages sent through the IM server of a large corporate department. In all 39740 messages from 223 senders were sent in a 72 day period. This data is a good representation of corporate use of IM. It may not be such a good representation of home use, but without any data it is hard to comment.

The throttle implements a rate limit on messages to "different" destinations, where "different" is determined as "not in a short recent history list". The algorithm is illustrated in Figure 4.

Every time a message is sent, the destination of the message is compared with the recent history list or working set. If the address is in the set, it is passed and if not that message is queued in the delay queue for sending later. At regular intervals messages are removed from the delay queue, their destinations added to the working set and the message passed. This mechanism ensures that the throttle allows a message to one new destination per time period.

Under normal operation, the majority of messages will be to destinations already in the working set, and only the occasional message will be delayed. If a virus attempts to spread, some of its messages will be passed (if they are to destinations in the working set), but others will be placed on the queue. The size of the queue can thus grow quickly (especially if the virus spreads much faster than the allowed rate) and be detected using a simple threshold. If the queue reaches the threshold it is pretty clear that a virus is attempting to spread and all further communication can be stopped.

The throttle thus has two responses: a delay to maintain the allowed rate, and a more severe block if the queue goes over the threshold. There are of course many variations possible, e.g. how the working set is maintained (a simple First-In-First-Out buffer, or using Least-Recently-Used replacement), whether the throttle allows credit i.e. free destinations if there has been no activity for a while [19], etc. It is also possible to run the throttle in "no-delay" mode, passing all the messages but updating the delay queue and working set as before. In this case the only response of the throttle is a block.

This algorithm would be best implemented at the messaging server, because that server either processes all the messages, or is responsible for the initiation of communication depending on configuration. Using the server also means that users would not be able to bypass the throttle. The server could be modified to run this throttle algorithm on behalf of each user of the server, holding messages to delay them, or refusing to accept messages in order to block the virus spread.

The first test is to see if IM traffic is suitable for throttling i.e. "to a slowly varying subset of buddies". To do this the throttle was simulated for each sender, and the total number of "different" recipients counted for different sized working sets. If this number is low, it shows that most messages are sent to destinations already within the working set. Figure 5 plots the results for all users who sent more than 100 messages, showing that the working set accounts for the large proportion of messages sent. For most of the users, a working set size of 5 gives less than 10% of messages to "different" destinations. It is important that this working set is kept as small as possible, as should a virus attempt to spread, messages to addresses in the working set are allowed without delay. The working set also forms a limit on the number of simultaneous conversations possible without delay. A value of 5 would appear to be reasonable in this regard.

There may be situations where this simple analysis does not hold. For example "group chat" applications where a message is sent to multiple recipients simultaneously. This was not used by any of the users in this dataset, but

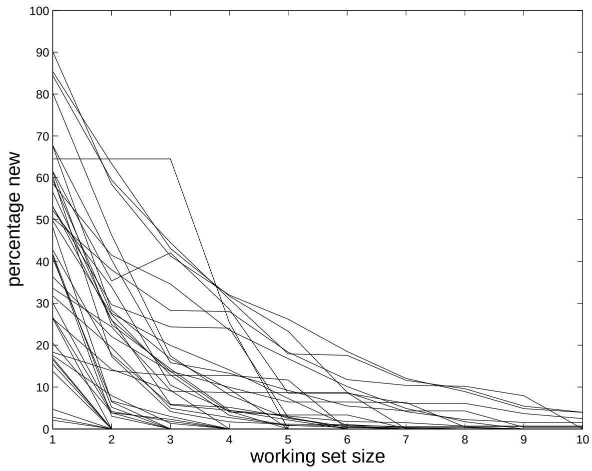


Figure 5: The number of "different" recipients for different sized working sets, with a different line for each user. A least-recently-used replacement strategy was used for the working set.

if it were, those messages would likely look to the throttle as if a virus were spreading. Some different mechanisms might be needed to handle this, but first some data on the usage of group chat would have to be collected. A similar situation occurs with email messages, and the design of the throttle for email has extra features to deal with multiple recipient emails. For details see [19].

The second major parameter of the throttle is the rate limit itself, i.e. how many new messages are allowed per minute, hour or day. Initial analysis shows that reasonable message delays are obtained if the allowed rate is around one new destination per minute. This makes intuitive sense: it takes time to compose, send, get the reply and send the next message.

Unfortunately, one of the ways for a virus to evade the throttle is to spread more slowly, e.g. sending messages every 2 minutes instead of every second. While this is an improvement (the loading on the messaging server would be much reduced) the speed of virus propagation would still be very fast. If a virus spreading at 1 message/second can infect the whole network in around 20 seconds, one sending messages every 2 minutes would take 40 minutes, which is still very fast. What is really needed is another order of magnitude, say a limit of 1 message/day. This would force the virus to spread so slowly (20 days to infect all) that other anti-virus mechanisms (patching, signatures) could be expected to work well.

One way to achieve this would be to run the throttle in "no-delay" mode, passing all messages but maintaining the delay queue as described above. For each message the size of the delay queue is checked, and if greater than a threshold then all further communication is stopped. In practise, all communication would be stopped while the user was warned and asked to vouch for the queued messages, perhaps by sending the user an instant message with the list of offending destinations. They could then override the server if this was a false positive (mistaken worm detection).

Since this version of the throttle works by calculating the rate of new addresses used, it would work independent of whether the messaging server was acting as a broker or a proxy. In the proxy case, the server would compare the destination address of each message with the working set maintained for each users, and update the delay queues, take action etc. if necessary. With the broker architecture, the same process could be used, the server checking working sets whenever users initiate chat sessions. To work well, this requires the IM client to not cache the connection details for a particularly long time, as if that were the case, the server would not see all attempts to chat with other users. However, given the time scales involved for the throttle (days), and given that the throttle limits messages to *different* destinations, the effect of any caching at the client will be minimal.

Figure 6 shows the maximum size of the delay queue (calculated as the number of unique recipients in the queue, not the number of messages) as a function of working set size for a throttle allowing 1 new recipient address per day.

The figure shows that even with this large time scale, remarkably small values e.g. working set size of 5, threshold of 2, would result in no warning for the majority of users. There are two or three users that would require higher values. This could be handled fairly automatically. Every user could be initialised with a default working set size of 5 and a threshold of 2. Every time a user goes over the threshold and it turns out to be normal usage, the size of that user's working set could be increased. Only a small proportion of users (2 or 3 out of 700) would be troubled by this learning process.

The low values for working set and thresholds are very good news for catching and stopping worms. It essentially means that if the virus sends to more than two destinations different from those in the working set over a 1 day period it will be detected and stopped. This will thus catch any virus sending messages at a higher rate than one per day, and will quickly catch a virus sending messages faster than this. To evade the throttle viruses would have to spread extremely slowly.

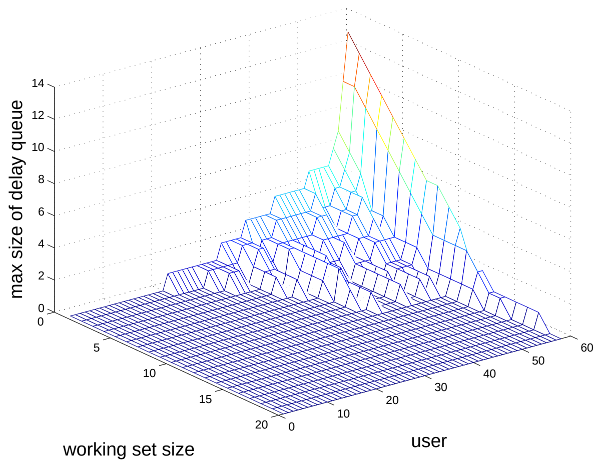


Figure 6: Maximum delay queue versus working set size for different users and an allowed rate of 1 recipient/day, with a least-recently-used replacement for the working set. The queue lengths for most users are low, even for small working sets. This means that the virus detection threshold could also be set low.

## 5 Effect of throttling on spread

Given that a certain set of throttle parameters is unlikely to interfere with the normal messaging behaviour of users, the question remains as to what effect this will have on the overall spread of the virus.

Since the virus detects when messages have been sent to too many different destinations, some messages will be sent before the throttle stops the virus. Those messages will infect other users and the virus will continue to spread. It is important that the number of escaped messages is small, as the more that escape, the larger and faster the infection will be. The primary effect of the throttle will be for the total number of infected machines to be smaller than without it.

When the virus infects a machine there is in general a fixed number of addresses (equal to the size of the buddy list $l$)[2] that it can use to spread. It chooses a recipient,

[2] It would be difficult for an IM worm to guess new addresses. This is because of the difficulty of guessing correct user names, and also because IM protocols only add addresses to a buddy list if both parties agree. This is to protect privacy. If A wants to add B to his list, the server will send a message to B saying "A wants to add your address

constructs the payload and sends the message. If the recipient matches the working set it will get passed, and if not, that recipient will be added to the delay queue. When the queue length reaches a threshold, the virus will be stopped. The question is: how many messages will escape before the queue length reaches the threshold?

If the threshold is $r$, the number of escaped messages is $r + k$ where $k$ is the number of recipients that the virus chooses that match the working set. $k$ must be less than the working set size $w$. This means that the fewest number of escapees will be $r$ ($k = 0$), and the most $r+w$ ($k = w$).

If the virus chooses sequentially from the buddy list, and the $w$ elements of the working set are distributed randomly throughout the buddy list, then the probability that $k$ will match the working set is

$$P(k) = \binom{r+k-1}{k}\binom{l-(r+k)}{w-k}/\binom{l}{w} \qquad (1)$$

where $\binom{a}{b}$ is the number of different ways to choose $b$ out of $a$ items, or $a!/b!(a-b)!$. This formula makes intuitive sense: the virus is stopped on its $r+k^{\text{th}}$ attempt, having found $k$ recipients in $r + k - 1$ elements. There are $\binom{r+k-1}{k}$ ways of doing that. The other $w - k$ entries must be in the rest of the buddy list i.e $\binom{l-(r+k)}{w-k}$, and the total number of ways of choosing $w$ in the buddy list is $\binom{l}{w}$.

Given this probability, the expected value of $k$ is

$$E(k|l, r, w) = \sum_{k=0}^{w} kP(k) \qquad (2)$$

making the total number of escaped messages

$$\begin{cases} l & \text{if } l < r + w \\ E(k|l, r, w) + r & \text{otherwise} \end{cases} \qquad (3)$$

Evaluating this for $w = 5$, $r = 2$ gives the behaviour shown in Figure 7. As the size of the buddy list increases, the number of escaped messages decrease. The best and worst cases are also plotted, showing that the number of escaped messages is likely to be small for reasonable sized buddy lists.

Going back to the topology described in Section 3, those users with the largest buddy lists are the ones that are most important for spreading the virus. This analysis shows that these are the ones that are likely to release the fewest messages before being throttled. This should have a profound effect on how the virus can spread.

to his buddy list", to which B replies.

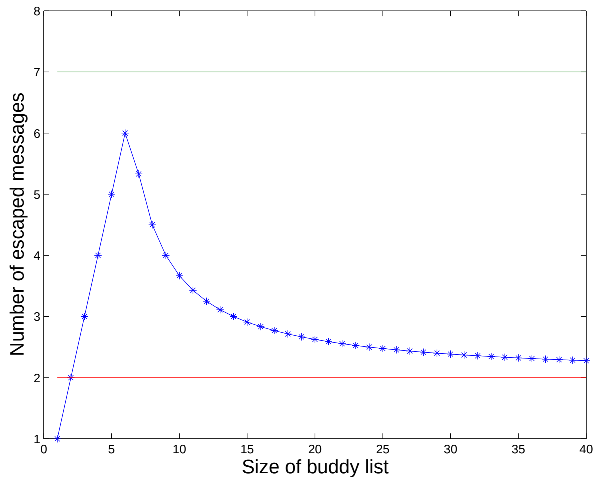


Figure 7: Expected number of escaped messages as a function of buddy list size for $w = 5, r = 2$. Also plotted are the best and worst cases (the two horizontal lines). The figure shows that if the buddy list is large, then the average number of escapees will be very close to the best case.

The effect of the throttle on virus propagation was simulated on the original network, using a slightly modified version from that described in Section 3. Four cases were calculated: no throttling; worst case ($r + w$ escapees); average case ($E(k|l, r, w)$ escapees) and best case ($r$ escapees). Figure 8 shows the results for the parameter setting above.

The best case outbreak is very small compared to without throttling, and the worst and average give outbreaks of around 2/3 and 1/3 of the unthrottled outbreak size respectively. This means that throttling has a significant impact on the spread of the viruses. The lower plot shows the effect on the traffic. The effect of the throttle is more strong here, with a large reduction in traffic in the average case.

## 6 Conclusion

Malware that spreads over instant messaging has the potential to cause a great deal of damage, and to cause that damage quickly. Current techniques to deal with such

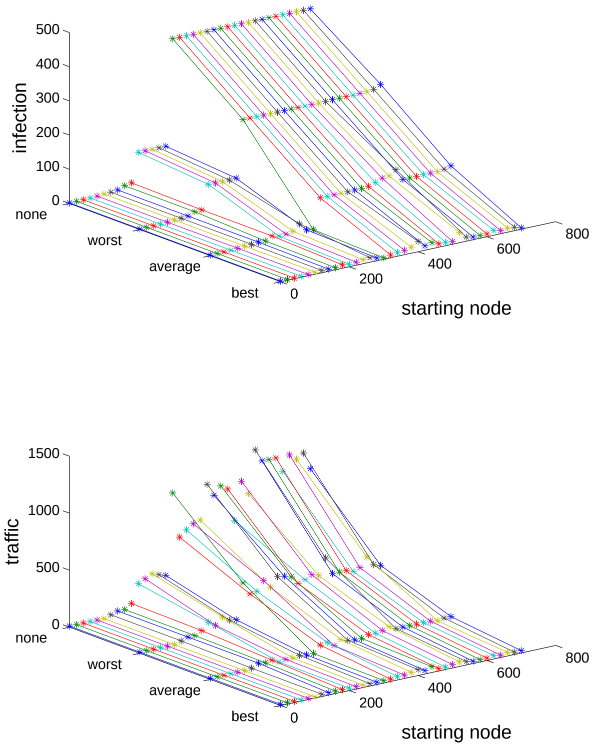


Figure 8: Effect of throttling on virus propagation. The upper plot shows the maximum size of the outbreak for different infection starting points, for four different conditions (no throttling, worst case, average case, best case). The lower plot shows the effect on the maximum traffic loading observed (messages/second).

threats are slow and reactive, and as such are likely to be ineffective [8].

This paper has presented a variation of Virus Throttling that can quickly detect when a worm or virus is sending messages over IM and stop the spreading further. This has been shown to not impact normal usage, and to be a sensitive detector of spreading malware. Throttling traffic in this way has a significant impact on both the size of the outbreak and the traffic generated.

The approach would be relatively straightforward to implement at the messaging server, and if used could drastically reduce the threats while maintaining the usability of instant messaging.

## References

[1] J. Balthrop, S. Forrest, M. E. J. Newman, and M. M. Williamson. Technological networks and the spread of computer viruses. *Science*, 304:527–529, Apr. 2004.

[2] A.-L. Barabási, R. Albert, and H. Jeong. Scale free characteristics of random networks: the topology of the world-wide web. *Physica A*, 281:69–77, 2000.

[3] V. Batagelj and A. Mrvar. Pajek – program for large network analysis. *Connections*, 21:47–57, 1998.

[4] CERT. CERT Advisory CA-2001-19 “Code Red” Worm Exploiting Buffer Overflow In IIS Indexing Service DLL, July 2001. Available at `http://www.cert.org/advisories/CA-2001-19.html`.

[5] Z. Deszö and A.-L. Barabási. Halting viruses in scale free networks, 2002. In cond-mat/0107420. Available from `http://www.arxiv.org/PS_cache/cond-mat/pdf/0107/0107420.pdf`.

[6] H. Ebel, L.-I. Mielsch, and S. Bornholdt. Scale free topology of email networks, 2002. In cond-mat/0201476. Available from `http://www.arxiv.org/PS_cache/cond-mat/pdf/0201/0201476.pdf`.

[7] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM*, pages 251–262, 1999.

[8] N. Hindocha and E. Chien. Malicious threats and vulnerabilities in instant messaging. In H. Martin, editor, *Proceedings 13th Virus Bulletin Conference*, pages 114–124, Toronto, CA, Sept. 2003. Virus Bulletin, Virus Bulletin.

[9] Jabber.org: Open source instant messaging, 2004. `www.jabber.org`.

[10] J. O. Kephart and S. R. White. Directed graph epidemiological models of computer viruses. In *Proceedings IEEE Symposium on Security and Privacy*, 1991.

[11] J. O. Kephart, S. R. White, and D. M. Chess. Computers and epidemiology. *IEEE Spectrum*, pages 20–26, May 1993.

[12] Microsoft windows update, 2004. `http://windowsupdate.microsoft.com/`.

[13] J. D. Murray. *Mathematical Biology, (2nd, corrected edition)*. Springer Verlag, New York, 1993.

[14] R. Pastor-Satorras and A. Vespignani. Epidemic spreading in scale-free networks. *Physical Review Letters*, 86(14):3200–3203, Apr. 2001.

[15] R. Pastor-Satorras and A. Vespignani. Immunization of complex networks. *Physical Review E*, 65:036104, 2002.

[16] J. Twycross and M. M. Williamson. Implementing and testing a virus throttle. In *Proceedings 12th USENIX Security Symposium, 2003*, pages 285–294, Washington DC, Aug. 2003. USENIX. Available from `http://www.hpl.hp.com/techreports/2003/HPL-2003-103.html`.

[17] D. J. Watts and S. H. Strogatz. Collective dynamics of small world networks. *Nature*, 393:440–442, 1998.

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

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