RESEARCH FEATURE 1

Virus Throttling

Matthew M. Williamson, Jamie Twycross, Jonathan Griffin and Andy Norman
Hewlett Packard Labs, UK

Virus throttling is a new technique to contain the damage caused by fast-spreading worms and viruses. Rather than attempting to prevent a machine from becoming infected, throttling prevents the virus spreading from an infected machine. This reduces damage because the virus is able to spread less quickly, and produces less network traffic.

Throttling is particularly effective against fast-spreading viruses, where signature-based approaches are weak. A signature-based anti-virus approach is really a race between the virus and the signature: a vulnerable machine will be infected if a virus reaches it before the signature does, but not if the signature gets there first.

Unfortunately, not only do modern viruses spread quickly, they also have a head start in the race as the result of any delay in generating the virus signature. In the case of these viruses, it is not just the infected machines that are a problem, the network loading caused by the additional traffic generated by the virus can cause problems for all users, not just those unfortunate enough to be infected.

Virus throttling is based on controlling an infected machine's network behaviour, and so does not rely on details of the specific virus – it does not need a signature. It restricts the virus spread, which is not as effective as preventing infection in the first place.

However, if it is impossible to prevent infection (i.e. the virus has reached the machine first), then restricting the spread of the virus will help contain the damage. The outbreak will grow less rapidly, because there will be fewer machines actively spreading the virus, and the network loading will be reduced.

By damping down the spread of the virus, throttling buys time for signature-based approaches, which are slower but more effective. Throttling makes the race more even.

How Does it Work?

Throttling relies on the difference between the network behaviour of a normal (uninfected) machine, and one that is infected by a virus. The fundamental behaviour of a virus is to replicate and spread itself to as many different machines as possible. For example, Nimda makes about 300–400 connections per second (cps) and SQLSlammer (see *VB*, this issue p.6) sends 850 packets per second, both probing for vulnerable machines. Many email viruses send mail to all the addresses they can find.

Our machines do not normally do this! They tend to contact machines at a much lower rate, and also contact the same machines repeatedly. The rate of connections to *new* machines is more in the order of one connection per second for TCP/UDP and once every ten minutes for email.

A virus throttle is a rate-limiter on interactions with new machines, where 'interactions' could be the initiation of a TCP connection, or the sending of a UDP packet or email, and 'new' is defined as the interaction having a different destination address from anywhere the machine has contacted recently. The throttle delays (as opposed to drops) connections that occur at a higher rate than that allowed.

If a virus attempts to scan for vulnerable machines at a high rate (e.g. 400 cps), the throttle will limit this to something much smaller (e.g. 1 cps). This will slow down the rate at which the virus can spread. If the virus is attempting 400 connections every second, and only one is being allowed, the backlog of delayed connections will grow rapidly. The length of this backlog is a reliable indicator that a virus has infected the system, and more drastic action can be taken. This involves preventing any further propagation e.g. stopping networking, and alerting IT staff.

The throttle thus slows down viruses until they are detected, at which point any further propagation can be stopped. For fast-spreading viruses, this whole process can take less than a second.

The rest of this article describes the throttling algorithm in more detail, discusses which protocols are suitable for throttling and presents some results from a user trial. The following sections then show how quickly the throttle can prevent onward propagation of the virus, as well as some experiments on how using throttles can affect the extent of a virus outbreak.

Throttle Algorithm

The throttle is a rate limiter on connections[1] to new machines, and is shown schematically in Figure 1. Whenever a request is made, the throttle checks to see whether the request is to a new host, by comparing the destination of the request with a short list of recent connections. The length of this list, or 'working set', can be varied – thus altering the sensitivity of the system. For example, if the length of the working set is 1, all requests other than consecutive connections to the same host will be 'new'.

If the host is not new, the request is processed as normal, but if it is new it is added to a 'delay queue' to await processing. Every time a timeout expires (indicated by 'clock' in Fig 1), the rate limiter process pops one request off the delay queue and processes it, thus ensuring that only one connection is made to a new host per timeout period.

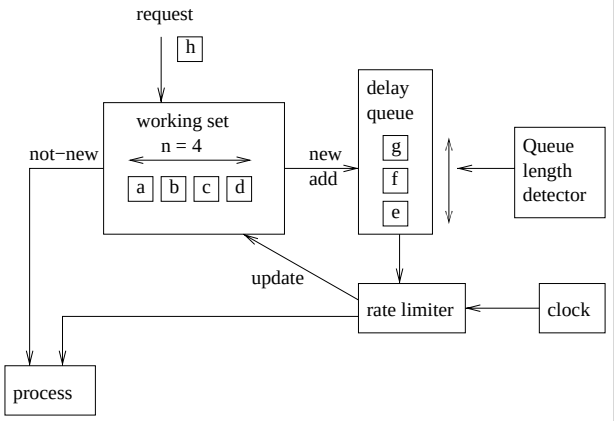

Figure 1: Virus throttling. When a request is made the destination is compared against a 'working set' of recently contacted hosts, to determine whether it is new or not. If new (e.g. the request is to host 'h', which is not in the working set), then the connection is queued in the delay queue. If not new, the request is processed as usual. The requests in the delay queue are popped off and processed at regular intervals, guaranteeing that no more than one connection per time interval is made to a 'new' host. Potential viral activity can be detected by monitoring the length of the delay queue.

The rate limiter processing involves releasing the request at the head of the queue and releasing any others to the same destination, as well as updating the working set with the new destination (removing a host from the working set and replacing it with the new destination).

Since the throttle implements a rate limit, and delays connections made at a higher rate rather than dropping them, if a process makes many connections they will mount up quickly in the delay queue. Therefore, monitoring the length of that queue gives a good indication of whether a process is acting like a virus. If the length of the queue reaches a threshold, the offending process can be halted either by stopping networking or by suspending the process itself. A user or administrator can then be contacted.

The allowed rate of connections to new machines is set to enable normal traffic to pass with minimal delay. If there are occasional periods when connections are made at a higher rate, these will be put on the queue, but the queue will not grow large, and so the delays will be small.

Since different protocols have different characteristics, it is possible to have a throttle per application, per protocol etc. The throttle itself could be implemented in a variety of places. On the host it could be inserted into the network stack e.g. as part of a software firewall, ethernet driver, etc., although being on the host makes it vulnerable to being switched off by a virus. It could also be implemented at various locations in the network. For protocols that use proxies e.g. email, web, etc. the throttling could be carried out at the server.

What Protocols can be Throttled?

In order for a machine to be 'throttled', its normal traffic must not look like a virus spreading i.e. connections made at a low rate, contacting the same machine repeatedly. The traffic for some protocols is nearly always to the same machine. For example SMTP (port 25), IMAP (143), web proxy (8088). Throttling on these protocols could be very tight since the destinations change so rarely. The behaviour for other protocols where the destination address changes is shown in Figure 2.

Normal network traffic was collected and the effect of the throttle was simulated for a range of working set sizes and allowed rates of connection. The average delay for each connection/interaction was then calculated. Figure 2 shows the different values of working set size and rate that give a constant average delay, for different protocols. The average delay for the TCP/UDP data is 0.3 ms, corresponding to three connections in every 1000 being delayed by one second.

The graph (Figure 2) shows that *Microsoft* file sharing (139), http (80), ssl (443) and dns (53) all look conducive to throttling, with a reasonable selection of working set sizes and allowed rates. The best settings are with a small working set and a low allowed rate, on the 'knee' of the curve.

Good values for http are thus a working set of 5, allowed rate 1 etc. Data for email (not shown) suggests that email can also be throttled, but the working set should be larger (around 20) and the allowed rate much lower (say one new email address every ten minutes).

Protocols that were found to be difficult to throttle were *Microsoft* naming (137) and another port used for file sharing (138 udp). The UDP traffic on both of these ports is to many different hosts at high rates, albeit in a bursty manner.

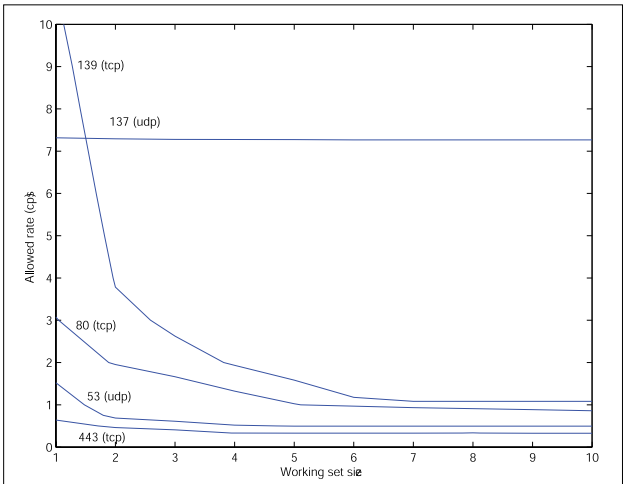

Figure 2: Throttle parameter settings for different protocols. The plot shows the settings for allowed rate and working set size that gives a constant average delay per connection. The different lines correspond to traffic with different destination ports. With the exception of UDP traffic on port 137, the other traffic is suitable for throttling, with different protocols requiring different parameter settings. The best parameter settings are with the smallest working set, and lowest allowed rate.

This data was collected for desktop machines, but one would expect similar patterns for servers (which primarily handle incoming connections and make outgoing connections to a limited number of machines).

There are notable exceptions: machines running scanners, web crawlers or notification services or, for UDP, a dns server. These would all look like machines infected with viruses (they have many interactions with different machines at a high rate).

It would thus be difficult to throttle such machines. However this is not the end of the world, it just means that, should a machine running one of these applications become infected with a virus spreading over the same protocol, then the spread from that machine could not be limited.

While the average delay gives some idea of what effect throttling would have on normal usage, the best test is whether those delays cause difficulties for users. One trial we have run involved throttling all TCP connections for three users over a two-month period. The working set size was five, and the allowed rate 1 cps.

Table 1 shows the results of the trial. 98% of connections occurred without delay, and the maximum delay was five seconds, occurring only once in over 80,000 connections. Anecdotally, the users did not notice any of these delays – the most likely explanation being that networks are full of delays of this sort of size.

Delay (s)	No. of requests	Percentage
0	80641	97.8%
1	1428	1.73 %
2	300	0.36%
3	29	0.03%
4	2	0.02%
5	1	0.01%

Table 1: Details of the user trial. For each delay (in seconds), the table shows the number and percentage of connections that were delayed.

To summarise, the analysis suggests that the majority of common protocols have normal traffic patterns that make them suitable for throttling. In addition, the sorts of delays that throttling creates in normal usage are small and not noticeable.

How Quickly does it Stop Onward Propagation?

As mentioned previously, the length of the delay queue can be monitored to detect if a machine is making many connections to many different machines and is thus likely to be infected by a virus. If the queue goes over a defined threshold, further propagation can be stopped.

Setting the delay queue threshold to 100 (fairly high, given that the queue never went above 5 in our user trial), Table 2 shows the time taken to stop virus propagation and the number of connections made for Nimda, and a number of different conditions for a hand-crafted 'test worm'.

Virus	Connections per second	Stopping time (s)	No. of connections allowed
Nimda	120	0.2	1
Test worm	2	106.1	104
Test worm	5	26.5	25
Test worm	10	11.2	11
Test worm	20	5.4	5
Test worm	40	2.3	2
Test worm	60	1.4	1
Test worm	80	1.0	1
Test worm	100	0.9	1
Test worm	150	0.2	0
Test worm	200	0.0	0

Table 2 – Showing time to stop and number of connections before stopping for a throttle with an allowed rate of 1 cps. Fast viruses are stopped very quickly (Nimda in 0.25 seconds), while slow ones are stopped fairly promptly (1.5 minutes for 2 cps). The virus is not able to make many connections before it is stopped.

The table shows some encouraging results, indicating that the faster the virus, the more quickly it is stopped, the time being less than a second for rates higher than 100 cps. Even viruses with a relatively slow connection rate are stopped quickly (just over 100 seconds for 2 cps). The number of connections that the virus is able to make is fairly small in all cases. The throttle is effectively preventing the onward propagation of the virus from the infected machine.

The fact that the throttle can detect the presence of a virus very quickly might make it useful as a monitoring device to provide early warning of viruses and collect data on their behaviour to enable quicker virus signatures.

Effect on Global Spread

A virus throttle is an altruistic idea – a throttled machine may still become infected, but it will not pass the infection on to others. Like most altruistic ideas, throttling will be most effective when it is widely used! Obviously throttling every machine is impossible in practice, so the question is: what is the effect on virus outbreaks if a smaller proportion of machines have the throttle?

That question is difficult to answer. There are many factors that make it hard to predict virus spread, let alone with throttling in the picture too. Virus spread depends on the propagation strategy of the virus, the density of vulnerable machines, the topology of the network, user behaviour, and so on. It also depends to a large extent on the whole process of fighting the virus: how quickly signatures are created and distributed, the vigilance of IT staff and so on.

To assess the impact of throttling, we tested the throttle against real viruses and measured spreading rates. Figure 3 shows the propagation of Nimda against time for an isolated subnet of 16 machines. When all the machines are throttled, Nimda does not spread at all. When no machines are throttled, Nimda takes about 20 minutes to infect all the machines. The other lines show that it is only when more than half of the machines are throttled that the infection is slowed, to about an hour when 75% of the machines have throttles.

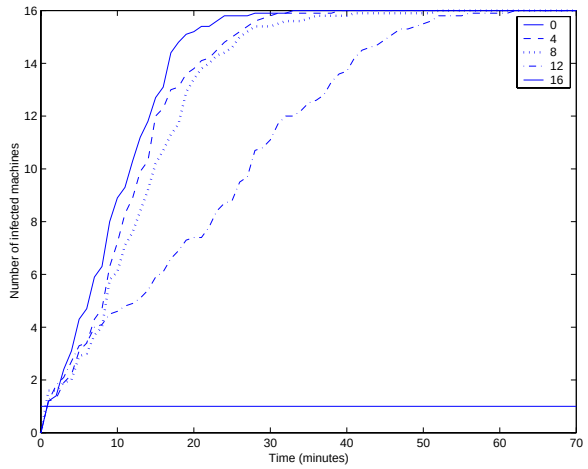

Figure 3: Number of infected machines plotted against time for the Nimda virus, varying the number of machines with throttles. Each line is the average of 10 runs. With no throttles, the virus infects all machines in about 20 minutes, but if all the machines use throttles, the virus does not spread at all. More than half of the machines need to have throttles in order to slow the virus significantly.

While the virus was slowed by a factor of three, it still spread through all the machines in one hour. This is quite a harsh test however, as Nimda's spreading strategy targets the local subnet, and so would be expected to spread most quickly there. However, the number of infected machines is not the only problem, the traffic they produce is important too. For the run with 12 of the 16 machines throttled, the traffic loading would be around one quarter of that in the unthrottled case, ignoring any saturation effects.

Part of the problem is that, because Nimda is a scanning virus, a single unthrottled machine can continue infecting others for as long as it is allowed to run. Other viruses do not scan (e.g. email viruses), and the throttle is likely to be more effective against those, mainly because the damage from an unthrottled machine will be much smaller.

We have also used modelling to look at throttling in the context of signature-based approaches, with a model of virus spread and clean up. The results from that model suggest that if 50–60% of machines (or more) have throttles, the impact of virus outbreaks can be much reduced[2].

Conclusion

Virus throttling is a new approach to containing the damage caused by fast-spreading worms and viruses. It targets the propagation of viruses from infected machines, slowing and stopping it. This reduces the number of machines actively spreading the virus, which in turn means that the outbreak grows more slowly, and the amount of traffic (and therefore disruption) is reduced.

Analysis of normal network traffic shows that the majority of protocols are suitable for throttling without interfering with normal usage, and that virus outbreaks can be reduced if a reasonable proportion (more than 60%) of machines are throttled.

Computer security is an arms race, with the attacking and defending technologies changing to exploit each other's weaknesses. Throttling is, in principle, quite hard to beat, because it targets a fundamental characteristic of a virus: its replication. It is not possible to replicate as a computer virus without contacting different machines!

On the other hand, one way to defeat the throttling technique would be to design slow viruses that pass through the limiter without delay. However, while the throttle would not stop such viruses, there are many other mechanisms already in place to deal with such slow threats.

A simpler way to defeat the throttle would be if malicious code were able to switch it off, as recent viruses (e.g. Bugbear) have begun to do with software firewalls and anti-virus software.

A virus switching off the throttle is worrying, because the throttle is designed to work after infection. There are two solutions for this, one is to hide and obfuscate the software on machines to make it more difficult to disable, and the alternative is to move the throttle into the network. Currently we are researching a variety of ways to do this. The throttle might already be implemented in the network;, for example the logical place for an email throttle is at the outgoing mail server.

In conclusion, throttling looks like a promising technique. The idea of concentrating on the possible harm that a machine can do, rather than on harm that could be done to it is a general and powerful one. If our machines were made so that they could cause less harm to one another, using throttling and other similar technologies[3], then our computer systems would be much more resilient when under attack.

Footnotes

1 The term 'connections' is meant to apply not just to TCP connections, but also to UDP packets and emails. The word should really be 'interactions', but that is clumsy and does not fit any of the protocols.

2 See 'An epidemiological model of virus spread and cleanup', M. M. Williamson and J. Leveille (2003), Submitted to USENIX Security Symposium 2003. This paper is available from http://www.hpl.hp.com/research/bicas.

3 See 'AngeL: a tool to disarm computer systems', D. Bruschi and E. Rosti, *Proceedings of the New Security Paradigms Workshop 2002*, pp.63–69.

Further Reading

M. M. Williamson (2002), 'Throttling Viruses: Restricting propagation to defeat malicious mobile code', *Proceedings of ACSAC Security Conference 2002*, pp.61–68, available from http://www.hpl.hp.com/techreports/2002/HPL-2002-172.html.

J. Twycross and M. M. Williamson (2003), 'Implementing and testing a virus throttle', Submitted to USENIX Security Symposium 2003, available from http://www.hpl.hp.com/research/bicas.

M. M. Williamson (2002), 'Resilient Infrastructure for Network Security', Submitted to "Complexity", available from http://www.hpl.hp.com/techreports/2002/HPL-2002-273.html.