

# Optimising Networks Against Malware

Pierre-Marc Bureau  
ESET, LLC  
1171 Orange Avenue  
Coronado, California, USA  
pbureau@eset.com

José M. Fernandez  
École Polytechnique de Montréal  
2500, chemin de Polytechnique  
Montréal, Québec, Canada  
jose.fernandez@polymtl.ca

## Abstract

*Rapidly-spreading malicious software is an important threat on today's computer networks. Most solutions that have been proposed to counter this threat are based on our ability to quickly detect the malware-generated traffic or the malware instances themselves, something that in many cases can be beyond our ability. Nonetheless, it seems intuitive that certain defensive postures adopted in configuring networks or machines can have a positive impact on countering malware, regardless of our ability to detect it. It is thus important to quantitatively understand how changes in design and deployment strategies can affect malware performance; only then does it become possible to make optimal decisions. To that purpose, we study in this paper the impact of network interconnection topologies on the propagation of malware. We first use a theoretical model based on Markov processes to try to predict the progression of an infection under varying interconnection scenarios. We then compare these predictions with experimental results obtained by launching a malware emulation agent on three differently configured networks. Both theoretical and experimental results provide quantitative confirmation of the intuition that networks with higher degrees of interconnection allow faster spread of malware. In addition to this, we believe that the models, experimental methodology and tools described here can be safely and fruitfully used to study other aspects of malware performance, and hence of the relative effectiveness of defensive counter-measures.*

## 1 Introduction and Background

In order to understand how to best defend against malicious software (*malware*), it is fundamental and necessary to a) understand and define performance criteria with which they are used by their creators and users, and to b) understand how various aspects of the environment in which malware operates can affect these performance criteria. Fur-

thermore, it has been argued [3] that it is also important to c) understand how changes in malware characteristics within a particular operating environment can improve or decrease its performance, because it allows us to conceive the worst possible threat (“optimal malware”) and hence try to prepare defences against it in advance. In this paper, however, we concentrate on the second aspect, i.e. understanding how performance of malware can be impacted by the operating environment with the ultimate objective of finding ways in which to “optimise” the operating environment, i.e. our networks, against malware of particular characteristics.

Of the various malware performance criteria that can be defined (stealth, footprint, damage inflicted, persistence, etc.) one of the easiest to quantify and measure from observed data is *speed of propagation*, i.e. the number of infected machines per unit of time. For that reason, various authors have studied the way malware, and more particularly computer worms, propagate inside a network.

Most modelling efforts in worm propagation used variants of the classical epidemiological model used in Immunology (see [4]). This model considers that a system can be in two states: *susceptible* or *infected*. The only possible transition between states occurs when a system goes from *susceptible* to *infected*, i.e. when a malware instance successfully infects it. This model is also called the *S* model because the curve representing the number of infected machines over time resembles the shape of this letter. Zou *et al.* [9] used an improved version of the classical epidemiological model that includes a variation of infection rate. Instead of using a constant to express the virulence of a computer worm, they proposed the use of a time-dependent function. This approach had the beneficial effect of showing that the spread of malware is slowed down by the amount of traffic it generates and decreasing the number of vulnerable hosts. Staniford *et al.* [7] modelled the spread of worms using a modified version of the epidemic model. With the help of this model, they predicted the appearance of improved worms with capacities to infect a vulnerable population in merely a few seconds. In addition to the classical

epidemiological model, Markov processes have been used [5] to construct analytical models of worm propagation and the impact that network topology has on them.

While some of the work above is inspired on in-the-wild observations of computer worms, it is hard to evaluate the validity of these models under varying malware and environment characteristics, a fundamental step for achieving the above declared environment optimisation objectives. The use of event simulators can partially address this issue, because it allows us to vary both malware characteristics and “experimental” conditions (i.e. characteristics of the environment). However, to this date only limited work has been done on worm propagation simulations. Of particular note is the work of Wagner *et al.* [8], that has used a custom-built simulator to study and predict the behaviour of self replicating code on a computer network. The simulator uses data gathered on peer-to-peer networks to consider the bandwidth available for groups of hosts. Their results thus take into consideration part of the network’s performance to study the speed of propagation of a computer worm.

The limitations of this kind of work on the study of worm propagation is three-fold. First, it is hard to adequately “calibrate” these theoretical models. One reason is that there is only limited historical quantitative data on in-the-wild worm epidemics with which to do so. Another is that the characteristics of historically observed worms are not identical. Secondly, most propagation data is of a global nature and seldom correlated with network connectivity, i.e. potential propagation paths. Consequently most models consider the cyber-universe (i.e. the Internet) as being homogeneous and isotropic (much akin to the way cosmologists regard the “real” Universe). As a consequence, such historical data cannot effectively be used to find out and confirm which network topologies are most effective in affecting worm propagation. Finally, whatever conclusions can be obtained from the analysis of historical worms, it is definitely not certain that these would still hold true for future worm infections portraying different characteristics.

In order to address these issues a methodological approach is required that combines both theoretical modelling, and *in vitro* experimentation that can validate theoretical predictions beyond the ranges of environmental and malware characteristics observed so far. In the rest of this paper, we will describe our initial efforts in trying to address this issue. To this effect, we have deliberately chosen to study the relatively well understood compromise between propagation speed and network filtering to illustrate and validate our approach. In Section 2, we describe both the particular type of worm infection and operating environment we have sought to model, as well as the Markov processes we have used to illustrate this compromise. Section 3 describes the experimental setup used to run malware experiments used to validate the model. In particular we describe the *Malware*

*Emulation Framework* we have created and the laboratory setup used. We compare the theoretical results obtained from modelling with those obtained from experimentation in Section 4. Finally, we discuss the limitations, advantages and prospects for future refinements of this approach to experimental research in malware in Section 5.

## 2 Propagation vs. Filtering

One of the ways in which we have intended our work to depart from previous malware research is in trying to put forth and propose methods for obtaining quantitative predictions about the performance of malware without having to depend exclusively on the analysis of historical data. In order to demonstrate them and to evaluate their usefulness, we have chosen to study one particular hypothesis concerning the propagation speed of a certain type of worm epidemic, within a specific type of network environment. In this section, we first describe the type of worm considered, and then discuss they type of network in which the hypothesis might be true. We finally describe the mathematical model used to validate this hypothesis.

### 2.1 Worm type and main hypothesis

In our work, consider worm epidemics of the “fire-and-forget” type, the prime example being the MS SQL Slammer worm [6], where individual malware instances have the same simple behaviour:

1. Choose at random a potential target within a given network address space  $S$ , of size  $|S| = s$ .
2. If the target is vulnerable then
  - (a) Infect the target by using one of the known vulnerable exploits,
  - (b) Copy the malware code to the target, and
  - (c) Remotely launch the malware code on the target machine.
3. Go back to Step 1.

Note now that we will hereforth only consider worms where all instances choose targets from within the same address space  $S$ . While this set of addresses is potentially very large, the chance that a suitable target is found in Step 1 depends, among other factors, on the proportion of vulnerable machines within the set of addresses being selected, but also on whether the chosen address is reachable through the network. Hence, it is intuitive to believe that such types of worms would have more difficulty to spread rapidly through networks with lesser degrees of connectivity between machines; this was the case for many well-known cases of

worm epidemics. This is no less than the cybernetic equivalent of the well known fact that human diseases spread more quickly in urban environments, where infected individuals come in contact with many more infectable individuals within a given time period than in rural environments.

While this intuition is not a new nor an earth-shattering discovery, we choose it as our test hypothesis for two good reasons. First, precisely because it is not new and non-controversial, it is a prime candidate for testing whether our quantitative methods are sound. Second, behind this hypothesis hides a telling compromise between security and performance. If it is true that malware propagation can be thwarted by tighter, less connected networks, is it worth it then to configure our networks accordingly? At the onset of the Internet, many of its applications had benefited from a virtually fully-connected mesh at the network (IP) and transport (TCP) layers. However, due mostly to the lack of IPv4 addresses, but also for security concerns, the use of private address spaces has become widespread for both corporate and home networks. These networks are now only accessible through gateways performing Network Address Translation (NAT) or through machines connected to both the public and private parts of these networks. Thus, worms such as those above would in principle have to first compromise a gateway or an interconnected machine in order to spread through these networks. In addition, the introduction of network filtering in its various forms (port filtering, egress filtering, etc.), by both host-based or network device-based, has further reduced the direct connectivity potential within the Internet.

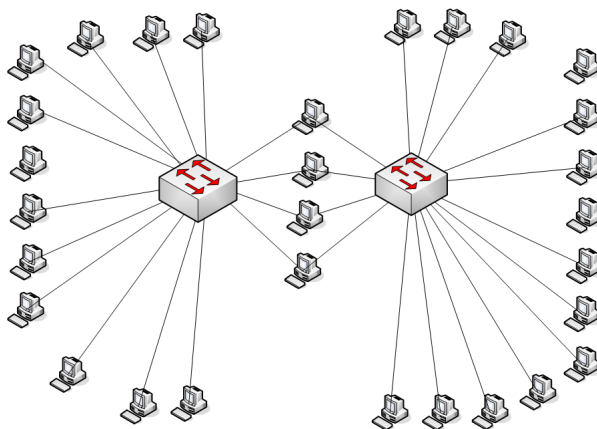
Unfortunately, the price to pay for this potential increase in security is a decrease of network performance and reliability. The necessary use of gateways or proxies adds latency to the end-to-end transmission times. In addition, they can also become bottlenecks of network traffic susceptible to congestion, and also single points of failure making networks vulnerable to fortuitous down time or even deliberate denial of service (DoS) attacks. Finally, as many system administrators and managers know, tight, locked down networks are seldom user friendly in that they often lack the flexibility required to support the ever-changing demands of the users (which, by the same token, arguably also makes very “administrator unfriendly”).

What is, then, the most acceptable compromise between security, on the one hand, and performance, reliability and flexibility on the other? In other words, what is the optimal “connectivity setting” that we should be configuring our networks at? This is a complex question whose answer depends on factors that go much beyond the scope of this work, i.e. the impact of connectivity on performance and reliability, and the quantification of the benefits of flexibility. However, one thing is sure, one cannot seriously try to address this question without considering the *quantifiable*

impact on security. This in itself is not a single question but many, as there are several (sometimes even conflicting!) security objectives. In this paper, we try to provide a small portion of the bigger equation by quantifying the impact of filtering on at least one security objective: slowing down worm propagation.

## 2.2 Network environment

In both models and experiments presented here, the network being represented consists of two subnetworks, with varying degrees of interconnection between them (see Figure 1). Even though all machines represented are vulnerable, this does not necessarily mean that all machines in the corresponding subnets *are* vulnerable. The presence of non-vulnerable machines can at most affect the rate of propagation by providing more addresses for the malware to try; since attempt on those machines will not result in infection, their presence slows down propagation. Ultimately, however, whether the addresses being scanned for vulnerabilities correspond to network-reachable but non-vulnerable machines, to non-reachable addresses or to non-existent machines, the impact on propagation will be the essentially the same. The reason is that the time it takes for the malware to determine that there is no vulnerable machines is roughly the same, i.e. the network round-trip time (RTT), for all of the above situations. Thus, our models and experimental results are without loss of generality in that regard.



**Figure 1. An example of a network with 30 vulnerable machines, with 13 machines in each subnetwork, and 4 interconnecting gateways.**

Note also that these “subnetworks” need not be IP subnets *per se*. Rather, they can represent sets of machines that malware instances can directly infect. For example, for worms propagating with a single UDP packet (e.g. SQL

Slammer) these represent vulnerable machines that are directly addressable. However, for malware using vulnerabilities at higher levels of the protocol stack these “subnetworks” could include machines that are not necessarily directly reachable by an IP packet but through other means. For example, penetration vectors using phishing emails (e.g. containing URL to malicious sites from which malicious code is downloaded towards to vulnerable clients) might bypass standard network filtering barriers, as both email and HTTP traffic would typically be forwarded through. In those cases, the network connectivity through a switch, such as represented in Figure 1, should be thought of instead as potential infection paths between machines.

Following the same reasoning, while for a SQL Slammer-type of worm, a “gateway” machine represents a machine that can receive and send the malicious packets on both subnetworks, this is not necessarily always the case. Again, if we think of these links as infection paths or *infection opportunities* between machines, a single machine interconnected to two such subnetworks could be instantiated in our models as more than one “machine” in the case where we are modelling a multi-vector worm. For example, malware running in a machine on the first subnetwork that is capable of exploiting two (or more) vulnerabilities, both present on a given, physical gateway machine, would have two different propagation paths towards such a gateway machine and therefore into the second subnetwork. Thus in Figure 1, the four “gateways” represented could in practice correspond to four different *services* on the same machine, all exhibiting vulnerabilities exploitable by the worm. A typical example of this situation would be a server in the DMZ of a small enterprise network accumulating functions as an SMTP server, Web server, HTTP proxy, etc. In other words, while the models and results presented here are expressed in terms of layer 3 network filtering, they can be extended to more general types of worms using other methods of propagation.

### 2.3 Markov process model

In order to build a model for the propagation of the worm, let us consider that the worm starts with a single machine in the first subnet, and let us call the  $I$  the set of infectable machines in that subnet. Similarly, we represent by  $J$  and  $K$  the set of vulnerable gateway machines and vulnerable machines in the second subnet, respectively. In the Markov process model we consider, a *state* of the system consists of a triplet  $(i, j, k)$ , where  $i, j$  and  $k$  describe the number of infected machines within each of the above sets  $I, J$  and  $K$ , respectively. As described, the initial state is thus  $(1, 0, 0)$ . Since we do not consider disinfections, machine deactivation or changes in network topology during the epidemic, the only non-*status quo* transitions possible

increase the number of infected machines. The various possible cases of infection, their necessary conditions and the corresponding state transitions are shown in Table 1.

**Table 1. Different cases of single machine infections**

Infect from	Conditions	State Transition
$I$ to $I$	$i <  I , i \geq 1$	$(i, j, k) \rightarrow (i + 1, j, k)$
$J$ to $I$	$i <  I , j \geq 1$	$(i, j, k) \rightarrow (i + 1, j, k)$
$I$ to $J$	$j <  J , i \geq 1$	$(i, j, k) \rightarrow (i, j + 1, k)$
$J$ to $J$	$j <  J , j \geq 1$	$(i, j, k) \rightarrow (i, j + 1, k)$
$K$ to $J$	$j <  J , k \geq 1$	$(i, j, k) \rightarrow (i, j + 1, k)$
$J$ to $K$	$k <  K , j \geq 1$	$(i, j, k) \rightarrow (i, j, k + 1)$
$K$ to $K$	$k <  K , k \geq 1$	$(i, j, k) \rightarrow (i, j, k + 1)$

Note that Table 1 does not include the possibility of multiple simultaneous infections. For example, we could have the case of multiple infections within the same subnet, such as the transition  $(i, j, k) \rightarrow (i + 2, j, k)$ , with  $i \leq I - 2$ , where the infecting machines could be either in  $I$  or  $J$ ; or the case of simultaneous infections in several subnets, such as  $(i, j, k) \rightarrow (i + 1, j + 1, k)$ , where in this case the infections could originate from  $I, J$  or  $K$ . In order to fully describe the Markov process, we need to complete Table 1 and compute transition probabilities for all possible state transitions.

Given an initial state  $(i, j, k)$ , let us denote as  $I', J'$  and  $K'$  the sets of infected machines within  $I, J$  and  $K$ , respectively. Furthermore, let  $\bar{I} = I \setminus I', \bar{J} = J \setminus J'$  and  $\bar{K} = K \setminus K'$ , be the sets of yet uninfected vulnerable machines; we also represent by  $\bar{i} = |\bar{I}| = |I| - i$ ,  $\bar{j} = |\bar{J}| = |J| - j$ , and  $\bar{k} = |\bar{K}| = |K| - k$ , the numbers of uninfected machines in  $I, J$  and  $K$ , respectively. The legal state transitions are thus:

$$(i, j, k) \longrightarrow (i + u, j + v, k + w)$$

where  $0 \leq u \leq \bar{i}$ ,  $0 \leq v \leq \bar{j}$ , and  $0 \leq w \leq \bar{k}$ , which we will represent for short as the event  $[+u, +v, +w]$  when the initial state  $(i, j, k)$  and the above conditions are explicit. To compute the probability of this generic event is relatively complex: we must partition each of the sets of infected machines  $I', J'$  and  $K'$  into separate subsets depending on whether the malware in those machines will propagate to uninfected machines in  $\bar{I}, \bar{J}$ , or  $\bar{K}$ , or not be successful in

propagating at all.

$$\Pr([+u, +v, +w]) = \frac{1}{s^{i+j+k}} \cdot \sum_{\substack{u = i_1 + j_1 \\ v = i_2 + j_2 + k_2 \\ w = j_3 + k_3}} \left[ \binom{i}{i_0, i_1, i_2} \binom{j}{j_0, j_1, j_2, j_3} \binom{k}{k_0, k_2, k_3} \right] \cdot \frac{\bar{i}!}{(\bar{i} - u)!} \frac{\bar{j}!}{(\bar{j} - v)!} \frac{\bar{k}!}{(\bar{k} - w)!} \cdot (s - \bar{i} - \bar{j})^{i_0} (s - \bar{i} - \bar{j} - \bar{k})^{j_0} (s - \bar{j} - \bar{k})^{k_0} \quad (1)$$

where the indices  $j_1, j_2, j_3$  represent the number of machines in  $J'$  that have infected machines in  $\bar{I}, \bar{J}$  and  $\bar{K}$ , respectively, and  $j_0$  the number of machines in  $J'$  that have not successfully infected anyone; and similarly for the  $i$  and  $k$  indices.

In our case, the Markovian property of the system, i.e the fact that the evolution of the epidemics at time  $t$  only depends on the current state  $(i, j, k)$ , is reasonable since all malware instances, as described above, act independently and within a memoryless behavioural loop. Under the assumption that all infecting machines are synchronised within a given time period  $\Delta t$ , in that each of them will be able to attempt exactly one infection in that period, then we can use the transition probabilities of Equation 1 to construct the stochastic matrix  $M$  of the corresponding Markov process. This is the case when the time it takes for an instance to copy itself and launch its copy (Step 2) is comparable to the time it takes it to discover that a target machine (chosen in Step 1) is not vulnerable or not reachable. We believe this is reasonable for small sizes of infection vectors, since the extra latency due to transmission and launching of code is probably a negligible compared to typical network RTT's and timeout values; in other words, the correct  $\Delta t$  for our model is in the scale of a few seconds.

If we represent the initial state  $(1, 0, 0)$  with a probability (column) vector  $\vec{p}_0$ , we can then compute the probability distribution of states at any future time. In particular, let  $\vec{p}_t$  represent the probability distribution of states at time  $t$  then, we can compute the probability distribution at time  $t + \Delta t$  as  $\vec{p}_{t+\Delta t} = M \cdot \vec{p}_t$ .

However, in order to achieve this goal, we have to evaluate the full expression of Equation 1 for every possible state transition, which is a very complex task for all but networks of just a few machines. Hence, we have simplified the model by ‘‘bundling’’ together all multiple transitions: whenever multiple infecting machines would concurrently be able to find suitable vulnerable machines, only one infection is registered. In other words, it is as if all infected machines could independently try to find a suitable target but only one of them was allowed to propagate. Notice

that this type of *tournament* propagation is different from *worm head* propagation, where only the last infected host is allowed to scan for new infected machines. Here, all infected machines are given the chance to find a suitable infected host, but only the ‘‘winner of the tournament’’ is allowed to propagate, within a given time interval  $\Delta t$ . With this type of propagation, the only cases of infections possible are precisely those of Table 1. It thus becomes much easier to compute the corresponding transition probabilities. For example, by instantiating Equation 1 for the transition  $(i, j, k) \rightarrow (i + 1, j, k)$ , we get a sum of two factors corresponding to the cases of the first two rows of Table 1:

$$\Pr([+1, 0, 0]) = \frac{1}{s^i s^j s^k} \cdot \left[ \binom{i}{1} \binom{j}{0} \binom{k}{0} \bar{i} (s - \bar{i} - \bar{j})^{i-1} (s - \bar{i} - \bar{j} - \bar{k})^j (s - \bar{j} - \bar{k})^k + \binom{i}{0} \binom{j}{1} \binom{k}{0} \bar{j} (s - \bar{i} - \bar{j})^i (s - \bar{i} - \bar{j} - \bar{k})^{j-1} (s - \bar{j} - \bar{k})^k \right] = \frac{1}{s^i s^j s^k} \cdot [i \bar{i} (s - \bar{i} - \bar{j})^{i-1} (s - \bar{i} - \bar{j} - \bar{k})^j (s - \bar{j} - \bar{k})^k + j \bar{j} (s - \bar{i} - \bar{j})^i (s - \bar{i} - \bar{j} - \bar{k})^{j-1} (s - \bar{j} - \bar{k})^k]$$

which for address spaces  $S$  much larger than the network sizes, i.e.  $s \gg |I|, |J|, |K|$ , can be approximated as:

$$\approx \frac{1}{s^i s^j s^k} [i \bar{i} s^{i-1} s^j s^k + j \bar{j} s^i s^{j-1} s^k] = \frac{1}{s} [i \bar{i} + j \bar{j}] = \frac{1}{s} [i(|I| - i) + j(|J| - j)] \quad (2)$$

and similarly for the transition events  $[0, +1, 0]$  and  $[0, 0, +1]$ .

With this simplification, it now becomes possible to numerically solve the full Markov model for networks of several tens (and even hundreds) of machines, as we did. In particular, we can track the predicted rate of propagation by plotting the evolution over time of the expected number of infected, i.e. the evolution of the value  $\bar{N}_t$ ,

$$\bar{N}_t = \sum_{(i,j,k)} \Pr((i, j, k) \text{ | at time } t) \cdot (i + j + k) \quad (3)$$

These plots are shown and compared with experimental results obtained in laboratory experiments in Figures 3, 4 and 5 in Section 4.

### 3 Experimental Methodology

#### 3.1 Constraints and requirements of malware experimentation

The prime considerations that should be addressed when performing experiments on malware is safety and ethics.

In a nutshell, the main problem is the following: how can one gain knowledge of what is inside Pandora’s box without opening it? The approaches that have been traditionally used by *bona fide* malware researchers (aka the “good guys”) to address this problem are the following:

**Historical analysis of *in vivo* infection data.** For important epidemics, some researchers directly or indirectly associated with government agencies or the anti-virus industry have successfully mined Internet-wide data sets containing information about the spread of computer worms. There are many technical problems associated with manipulating and analysing such data, and these are often of limited availability. In addition, they are of limited use for the kind of research that we are interested in, i.e. understanding the compromises between malware performance on one side, and system configuration and performance of defensive systems on the other, since it is often very hard to correlate the collected data with actual network and system configurations.

***In vitro* execution of wild malware.** Execution of existing malware captured in the wild, in a controlled and isolated laboratory environment, partially averts this last problem because the environment parameters can be altered in a controlled fashion. Still, this approach has serious drawbacks. Typical laboratory environments used for this kind of experiments are limited in size and flexibility, and thus only thin slices of the configuration space can be explored. Most importantly, the validity of the results is limited to known threats or future threats of similar characteristics.

***In vitro* execution of new synthetic malware.** One way to address this shortcoming is to use synthetic malware, built either by modifying existing wild malware or by creating new code, that explores those parts of the feature space that wild malware has not yet explored or not done so fully. This approach is very controversial and is even openly discouraged by the anti-virus community. The main reasoning behind this position is that despite the extensive technical and procedural safety measures that may be put in place, the risk is too great for such synthetic malware to find its way, whether accidentally or deliberately, in real systems.

In our work, we have sought to devise an experimental approach with the flexibility necessary to explore both the malware feature space and the network and system configuration space in a way that would allow us to draw sufficiently general conclusions. This, while at the same time adequately reducing risk and addressing ethical concerns. In addition, in order to facilitate the application of sound scientific principles, we have incorporated in our solution features that enhance and facilitate the gathering of experimental data and experiment control, tracking and repeatability.

### 3.2 Malware Emulation Framework (MEF)

To develop a flexible approach to explore both malware feature space and system configuration space, we have created the Malware Emulation Framework (MEF). The MEF is a tool for creating mobile agents that emulate the behaviour of malware; it is *not* a malware development toolkit. The MEF is a set of Ruby classes that lets the researcher organise the malware characteristics to be emulated according to the OODA loop [3]. The agents created are Ruby scripts, that are then launched on a safe, isolated *in-vitro* environment, much like has been done with wild malware experimentation. The MEF-generated agents all include a set of desired characteristics and a component for *telemetry*. The telemetry component makes the malware communicate with a telemetry server every time it executes an iteration of the OODA loop. The telemetry server also sends information to the agent to instruct it to either continue its execution or to stop. Telemetry thus lets the researcher gather important information on the spread of emulation agents inside a network and control the experiment by stopping it in the event of a problem. In addition, the telemetry code is used for experiment control and safety: if the telemetry server is not reachable, all agents immediately terminate their execution; this feature can be used to stop and restart experiments, by simply disabling the server.

The MEF is quite flexible and easily allows the inclusion of new behavioural features into the agents’ behaviour; each new behaviour is encapsulated in a new Ruby class, with a common interface with which it is referenced and activated by the OODA-loop engine. Certain basic behavioural features related to the Observation phase have already been implemented and are part of the MEF base classes. Network monitoring features of both the passive (PCAP-based) and active types (programmable TCP and UDP network scans) have been implemented. A simple command-and-control and communication feature has also been implemented using IRC that allows agents to communicate between themselves, *a la* botnet.

Another important feature of the MEF is its ability to integrate Metasploit exploit code [1]. While this feature was not used in the experiments reported here, this potentially allows the use of MEF to test the behaviour and performance of defensive systems in a much wider variety of operating environments. The integration into MEF of readily-available exploit code in Metasploit format provides an efficient method for evaluating the performance impact of changes in many malware features, such as type of shell code, communication methods, and type of exploit vector (client-side, remote buffer overflows, etc.). Furthermore, it can be used on a variety of target OS (Windows, Linux, Mac OS X, BSD), for all of which there are readily available

Ruby interpreters. By the same token, the need for a Ruby interpreter to be installed is arguably a “safety feature” of the MEF in itself (albeit, definitely not the only one), as it is very seldom present in production machines.

It is important to note that the MEF architecture has been deliberately designed and structured for code cleanliness and flexibility, not for efficiency. This in part due to the fact that for the evaluation of many malware performance criteria lack of execution-time speed is not critical; it can even be compensated for in laboratory conditions. Secondly, it makes it less likely that it could be used as a basis for wild malware development.

In summary, the MEF offers a very interesting compromise between the *in vitro* use of wild malware and that of synthetic malware for research purposes, keeping much of the flexibility of the latter while avoiding to enter the safety and ethical quicksand associated with its creation. In addition, its built-in telemetry and control features allow simpler and more flexible laboratory protocols than for experiments using wild unadulterated malware.

### 3.3 Testing Environment

The other side of the malware experimentation problem is the ability to construct a safe yet realistic and flexible operating environment on which to execute malware. Furthermore, this environment must have the ability to let us make non-intrusive observations, in a way that minimises their impact on experimental results. Many solutions to this problem have been proposed, including 100% hardware-based laboratory networks, virtual environments, and hardware- and software-based network simulators. Each of these methods has its own advantages and inconvenients. In the context of this work, however, we have chosen to setup our operating environment using virtualisation technology to emulate both systems and networks. On the one hand, with the limited resources available at the time, hardware-based solutions would have only allowed us to explore very limited network sizes (5-10 machines at most). On the other, the granularity of network simulators does not account for host-level behaviour and would not have allowed to actually emulate malware behaviour as described in Section 3.2.

More concretely, we have used the VMware ESX Server (version 3) virtualisation suite [2] to emulate a network of up to 30 machines on a single dedicated high-performance server. The server we used had two AMD Opteron processor with 2 GHz clock speed and 8 Gb of RAM. The virtual network emulated and used for our experiments is shown in Figure 2. It incorporates *experiment* or *test networks* on which all emulated malicious traffic travels, and a separate *control network*, all implemented as separate VMnets in VMware. The control network is used to automatically

configure the systems, send and receive telemetry information, start and stop the experiment, and clean the systems after the experiment.

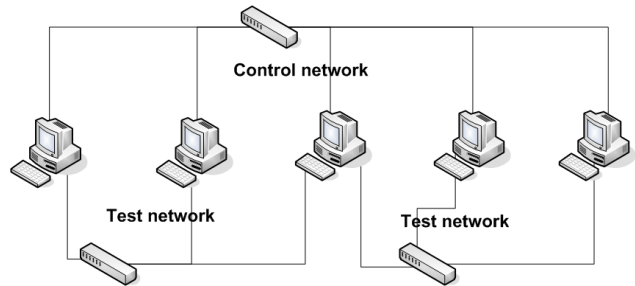


Figure 2. Virtual experimental network

The virtual machines used are based on the Debian distribution of Linux. Each machine has three network interfaces, two of them can be configured in an *ad-hoc* fashion depending on the experimental scenario, and the third one has a fixed address for the control network. Each virtual machine uses 32 megabytes of RAM, has a Ruby interpreter and runs a secure shell (SSH) server. The SSH server is used both to configure the hosts through the control network interface, and also as an “infection vector” on the experiment network interfaces used by the malware emulation agents to spread.

One of the important advantages of such a testing architecture is that even in the case of virulent and massive epidemics, telemetry data can still be reliably collected on the control network, without affecting or being affected by malicious activity and related traffic. However, while the control network might not be congested in such circumstances, degrading performance of both the virtual machines and virtual switches due to time-sharing with the emulated malware processes might affect the arrival times of telemetry events at the server. Nonetheless, close monitoring of virtual machines and real hardware performance data can be used to verify that this is not the case. In the experiments described in the next section, we used the VMware usage statistics export-to-Excel feature to verify that virtual machine CPU usage was never over 40%. Finally, the use of a separate control network provides an efficient and elegant path for configuring and cleaning up hosts after each experiment. This is in part made possible and greatly simplified by the virtual machine image snapshot and rollback features of the VMware software.

## 4 Experimental Setup and Results

The results presented in this section are two-fold. On the theoretical side, we used the Markov model derived in Section 2, to obtain predictions for the expected number of

infected hosts (Equation 3). A simple custom-built program was used to construct the state transition graph, calculate the corresponding probabilities by using the approximations of Equation 2, and plot the expected number of infected hosts. On the experimental side, we used a malware emulation agent built with MEF, and then launched it on one of the virtual machines of the virtual experiment network described above. This agent uses no real exploits to propagate, but instead is provided with client-side private keys for which the corresponding public key certificates are present in the database of SSH servers running on all machines on the network. In other words, the infection vector is emulated by opening a “backdoor” on the SSH server. Described in terms of the OODA loop, the malware agent has the following characteristics:

- **Observation.** Host discovery: by attempting an SSH connection on a IP address pseudo-randomly generated from within its own class C (/24) subnet, i.e it chooses one of 254 possible addresses.
- **Orientation.** None.
- **Decision.** Rule based: IF selected machine is “vulnerable” THEN infect & propagate.
- **Action.** These are the possible actions the agent can take:
  1. *Infection:* By successfully establishing an SSH session with vulnerable machine using hard-coded client-side private key.
  2. *Propagation:* By copying and executing the agent’s Ruby script through the SSH channel opened during the infection phase.

Three separate scenarios were used in both theoretical modelling and experimentation, in order to explore network configurations with increasing degrees of network filtering.

- **Scenario 1.** Corresponds to a network of 30 completely interconnected machines; this is, any of these 30 machines can reach any of the others. In terms of theoretical modelling, this means that the sets  $J$  and  $K$  are empty, with  $|I| = 30$ ; the final all-infected state is thus  $(30, 0, 0)$ . Experimentally, all 30 machines are connected to the same VMnet.
- **Scenario 2.** Corresponds to two separate subnetworks of 13 machines each ( $|I| = |K| = 13$ ), and four interconnected machines ( $|J| = 4$ ); the final all-infected state is  $(13, 4, 13)$ . This was emulated by setting up two different VMnets (in addition to the control network), connecting 13 machines to the first, 13 to the second, and four others to both of them.

- **Scenario 3.** Same as Scenario 2 but with only one machine interconnected to both experiment networks, and 14 machines in the first subnet. The final all-infected state is thus  $(14, 1, 15)$ .

It is important to note that the SSH connection timeouts were set at their default values of approximately 1 second, which meant that it would take that long for the emulation agent to detect that a machine was not vulnerable (in this case, not present at all). Similarly, when the guessed IP address corresponded to a machine present on the network, the time it would take for the agent to establish a connection, copy itself and remotely launch the copied code on the infected machine was consistently just under a second. For that reason, the  $\Delta t$  time step for the theoretical model was set at exactly 1 second. The numerical results obtained with theoretical modelling are shown in Table 2. The time values shown are the time in seconds (in this case corresponding to the number of steps) that it takes for the expected number of systems of Equation 3 to reach the number of systems shown. Since in the theoretical model the all-infected state never has probability one, we chose 29.90 as our “threshold” for complete infection, corresponding to a 99.5% probability of total infection. As can be seen, theoretical modelling already indicates what we suspected: greater levels of filtering reduce the rate of infection.

**Table 2. Numerical results for theoretical model.**

Expect number infected syst.	Time to infection (s)		
	Scenario 1	Scenario 2	Scenario 3
10	140	224	268
20	252	388	492
25	320	504	652
29.90	532	944	1208

Experimental results were obtained by running 5 differently seeded emulation runs for each of the three scenarios. The data collected by the telemetry server allowed us to determine precisely when each machine was infected and construct infection traces accordingly. The average time to infection for various number of infected systems thresholds, the shortest and longest times for total system infection, and the overall average of the variance are shown in Table 4.

The high level of statistical variance observed in Scenarios 2 and 3 is not surprising. For most infection runs, a plateau is reached when most or all of the machines in the first subnet are infected, with the “tipping point” being the infection of the first (or sole) interconnected machine. The times at which this event happened varied widely in the different runs, which is natural as the individual probabil-

**Table 3. Results of emulation experiments**

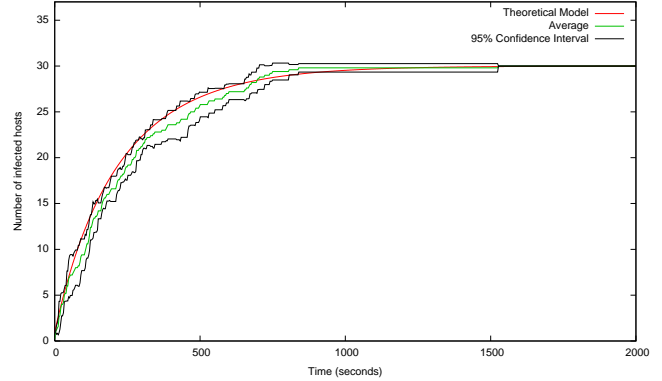
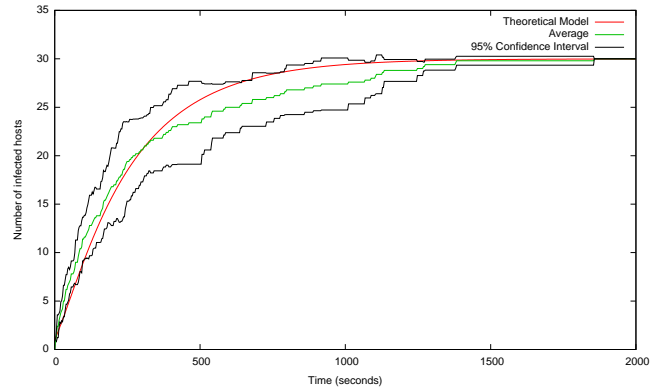
Time in (s)	Scen. 1	Scen. 2	Scen. 3
Average variance (5 runs)	0.5687	3.87	10.82
Shortest to infect 30 syst.	696	1107	771
Longest to infect 30 syst.	1527	1857	2094
Average to infect 30 syst.	903.6	1400	1377
Average to infect 25 syst.	483	639	969
Average to infect 20 syst.	279	279	471
Average to infect 10 syst.	108	87	141

ity  $p$  that an infected machine succeeds in infecting another in  $\Delta t$  is relatively low, i.e.  $p \leq 14/254 \approx 5\%$ . A second plateau with high variance is also reached for the infection of the last machine. Hence, it is more significant to look at the mean times of infection of intermediate values (25, 20, 10 machines, etc.). In fact, for almost all these intermediate threshold values, the mean time to infection confirms the hypothesis that higher levels of filtering slows down propagation, with lowest times in Scenario 1 (no filtering) and higher times in Scenario 3 (a single gateway). The only exceptions are the mean time to total infection, which was observed to be slightly larger in Scenario 2 than in Scenario 3, and the mean time for 10 infections, which was faster in Scenario 2 than Scenario 1; again we do not believe this to be statistically significant given the relatively small number of runs and the high variances observed.

The comparison between the theoretical and experimental results is shown in Figures 3, 4 and 5, for Scenarios 1, 2 and 3, respectively. The fact that we compute probabilities according to Equation 2 means in essence that multiple transitions are counted as single transitions. Thus, in principle the model should predict slower propagation times than those observed in experimental runs, where those multiple transitions can and do happen. We did not expect this simplification to make a significant difference, as it is equivalent to a first-order approximation of Equation 1 with respect to  $p$ , the infection success probability of individual machines: with  $p \leq 0.06$  and the leading terms in the multiple transition probabilities being  $O(p^2)$ , these terms are relatively very small. Nonetheless, this under counting effect is somewhat noticeable at the beginning of the infection runs for Scenarios 2 and 3, with the theoretical predictions “catching up” later in the infection run. Nonetheless, in all cases the theoretical predictions are relatively close to the experimental results, staying almost entirely within the confidence intervals at 95%, as shown in Figures 3, 4 and 5.

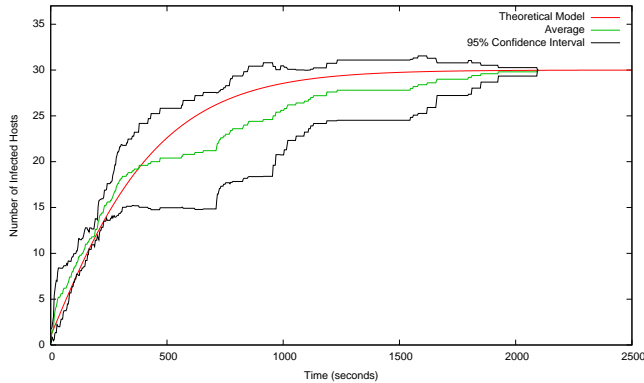
## 5 Conclusions

In this paper, we have sought to describe and introduce methods of both theoretical modelling and controlled ex-

**Figure 3. Theoretical and experimental results for scenario 1****Figure 4. Theoretical and experimental results for scenario 2**

perimentation that can be brought to bear to answer questions on the relative performance of malware on different operating environments. Such efforts are crucial in developing sound best practices and security engineering principles that will allow us to most efficiently configure our systems and networks, and most effectively construct and employ counter-measures against both known and unknown threats.

To illustrate our approach, we have chosen a test hypothesis on the compromise between network filtering and the rate of propagation of a certain kind of uncoordinated computer worm epidemics, of the type that has been often observed in the last few years. On the theoretical side, we have constructed a model based on Markov processes for which we have derived state transition probability equations for simple networks. These equations are relatively complex to solve, but a simplified approximation of the model nonetheless provides adequate predictions quite close to observed experimental results. On the experimental side, we have developed the Malware Emulation Framework (MEF)



**Figure 5. Theoretical and experimental results for scenario 3**

that allow us to quickly create prototype mobile agents that reproduce the behavioural characteristics of malware in a controlled laboratory environment, while avoiding many of the limitations and pitfalls of experimentation with real malware, whether wild or synthetic. We have also successfully used a 100% virtual environment, with a dual and separated control and experiment networks, to successfully run experiments on networks of up to 30 machines.

In summary, both theoretical and experimental results largely confirm the hypothesis that higher levels of filtering consistently and significantly decrease the rate of infection on small LANs, within the parameters observed. One of the benefits of introducing filtering is to increase the time to infection of various parts of the network: by up to 50% and by up to 100%, by reducing the number of interconnected machines to four and one, respectively.

We believe that both the theoretical and experimental methodologies introduced here set the standard for a new kind of solid quantitative research in the area of malware performance, but most importantly, for that on the performance of defensive counter-measures and the effectiveness of various configurations and procedural defences. Nonetheless, the test hypothesis we have employed to illustrate and validate these methods is of limited usefulness *as is*. Further research is needed to optimise the theoretical computations such that predictions can be made for more interesting sizes of organisational and corporate networks, such as of several thousands or tens of thousands of machines. Similarly, research facilities that are being built at the authors' university will allow for experiments such as those described above involving networks of several thousand machines. Complementary to this, and in order to hope to *optimise* our networks in terms of both security and performance, the quantitative relations between filtering and propagation speed obtained here should be contrasted with the similar quantitative relations concerning filtering and

network performance and reliability. Finally, we would also like to use this approach to discover and confirm similar compromises concerning other malware performance criteria and other types malware characteristics. All of these questions are the object of ongoing research by the authors.

## Acknowledgements

Most of this work was done while PMB was still a graduate student at the École Polytechnique. PMB wishes to thank his current employer ESET LLC, an anti-virus company, for allowing him to continue to work on this research and this paper. Both authors wish to thank Giuliano Antoniol for allowing the use of his laboratory for the experiments, and Louis-Alexandre Leclaire for his invaluable technical assistants. We also thank Francois-Raymond Boyer and Michel Dagenais for fruitful discussions and many suggestions for improvements.

## References

- [1] The Metasploit Project .  
<http://www.metasploit.com>, 2003.
- [2] VMware ESX Server Web page .  
<http://www.vmware.com/products/vi/esx>, 2007.
- [3] P.-M. Bureau and J. M. Fernandez. Optimising malware. In *Proc. IEEE International Swarm Intelligence and Other Forms of Malware Workshop (MALWARE)*, pages 34–45, 2006.
- [4] J. Frauenthal. *Mathematical Modeling in Epidemiology*. Springer-Verlag, New-York, USA, 1980.
- [5] M. Garetto, W. Gong, and D. Towsley. Modeling malware spreading dynamics. In *Proc. IEEE Conference on Computer Communications (INFOCOM)*, pages 1869–1879, 2003.
- [6] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the Slammer worm. *IEEE Security and Privacy*, 1(4):33–39, 2003.
- [7] S. Staniford, V. Paxson, and N. Weaver. How to own the internet in your spare time. In *Proc. USENIX Security Symposium*, pages 149–167, 2002.
- [8] A. Wagner, T. Übendorfer, B. Plattner, and R. Hiestand. Experiences with worm propagation simulations. In *Proc. ACM Workshop on Rapid Malcode (WORM)*, pages 34–41, 2003.
- [9] C. C. Zou, W. Gong, and D. Towsley. Code red worm propagation modeling and analysis. In *Proc. ACM Conference on Computer and Communications Security (CCS)*, pages 138–147, 2002.