# Overcoming Limitations in Computer Worm Models

by

Frank S Posluszny III

A Thesis

Submitted to the Faculty of

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

January 2005

APPROVED:

Professor Fernando C Colón Osorio, Thesis Advisor

Professor Micha Hofri, Thesis Reader

Professor Michael Gennert, Head of Department

## Abstract

In less than two decades, destruction and abuse caused by computer viruses and worms have grown from an anomaly to an everyday occurrence. In recent years, the Computer Emergency Response Team [cer04] has recorded a steady increase in software defects and vulnerabilities, similar to those exploited by the Slammer and Code Red worms.

In response to such a threat, the academic community has started a set of research projects seeking to understand worm behavior through creation of highly theoretical and generalized models. Staniford *et. al.* created a model to explain the propagation behaviors of such worms in computer network environments. Their model makes use of the Kermack-McKendrick biological model of propagation as applied to digital systems. Liljenstam *et. al.* add a spatial perspective to this model, varying the infection rate by the scanning worms' source and destination groups. These models have been shown to describe generic Internet-scale behavior. However, they are lacking from a localized (campus-scale) network perspective.

We make the claim that certain real-world constraints, such as bandwidth and heterogeneity of hosts, affect the propagation of worms and thus should not be ignored when creating models for analysis. In setting up a testing environment for this hypothesis, we have identified areas that need further work in the computer worm research community. These include availability of real-world data, a generalized and behaviorally complete worm model, and packet-based simulations. The major contributions of this thesis involve a parameterized, algorithmic worm model, an openly available worm simulation package (based on SSFNet and SSF.App.Worm), analysis of test results showing justification to our claim, and suggested future directions.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

A computer virus is a program that propagates itself by injecting its own code into another program or file. As this host file is shared among friends, colleagues, and other individuals, the virus is carried along with it. When the host is executed or viewed, the virus code is executed as well and finds other host files within the new system. In this way, the virus itself has no direct control over its propagation.

A computer worm, on the other hand, is a self-contained program whose main purpose is to propagate (copy itself)[wor04]. The 'host' for a worm becomes an individual machine. However, unlike a virus, the worm has choice over how it propagates to other systems, though it still depends on having its code executed on a remote machine through some means. Some worms take advantage of humans to facilitate propagation, through such simple tasks as reading e-mail or playing a downloaded game. However, a worm may spread more rapidly, and thus will be more dangerous, if it exploits a vulnerability in the network or system it is attacking as a vehicle for replication. The worm may take advantage of such a vulnerability and execute its code remotely, without any human interaction. One real-world example of such a worm is the *SQL Slammer*, which hit the Internet in January of 2003 [MPS+03]. Slammer caused Internet-scale congestion, effectively shutting down many e-commerce websites, ATMs, and the networks of some federal agencies [sla04]. In this thesis, and due to the significant damage potential of Internet worms, we focus our attention solely on worms and worm behavior and ignore viruses.

In general, a worm can be considered to contain two parts or main functions. These are: (1) the propagation mechanism used, and (2) a set of actions, usually referred to as "everything else." The propagation mechanism refers to the method used by the worm to infect vulnerable hosts. Examples of a worm's propagation mechanism include sending itself as an e-mail attachment to everyone on your contacts list or exploiting a vulnerability in a common file-sharing program. All of the other set of actions that a worm may take, commonly called the payload, is composed of everything the worm does besides propagate. Some worms will simply propagate and do not contain any payload. In other instances, worms have been found that perform Denial of Service (DoS) attacks against specific websites, install trojan or back-door programs on the infected system, and even un-install other worms [Inc04a, Inc04b, Inc03]. We believe that these other actions may be further broken down into a self-preservation module and other goal-based actions (GBA) modules. Self-preservation could be as simple as disabling anti-virus software or adding itself to the host machine's startup scripts. GBAs are included in the worm based upon the authors ordered set of goals. Further discussion of this may be found in Section 4.

As worms become more prolific and carry more dangerous payloads, there becomes an obvious need for more accurate and timely detection and prediction methods. Detection methods may be used to help mitigate an oncoming or ongoing attack. Predictive methods may be used to forecast how a theorized or recently discovered worm will act *in the wild*, unrestrained on the Internet. They may also be used to calculate limits on what is possible for a worm to accomplish, which may be utilized in such a sector as Electronic Insurance where the probability of a worm affecting a particular network and causing damage can be calculated. Known worms are detectable with today's signature-based anti-virus tools, however these methods are based upon knowing specifics of what the particular worm looks like, thus the need for updating worm libraries or databases whenever a new threat emerges. If one were to look not at what a particular worm's code is, but rather at the behavior of the program, then one would be able to detect "new" worms through their actions. We choose to focus on this behavioral-detection, specifically with respect to propagation from host-to-host within a localized network. While other behavioral aspects may provide useful detection data as well (such as actions taken within a host or resource utilization), the study of such behaviors is outside the scope of this thesis.

The remainder of this thesis is organized as follows. Section 2 provides background material to help in understanding worm propagation and the previous work that has been accomplished in this area. Section 3 describes our approach to testing the hypothesis, including initial problems encountered and the final directions taken. The design of a parameterized, algorithmic worm model is covered in Section 4. Simulations are described in Section 5, with testing, results and analysis in Section 6. A summary and final conclusions are given in Section 7, along with suggested directions for future work. Appendices follow with more detail explanations of the simulation design and usage.

# 2 Background

This section describes the concepts of computer worms and their network propagation, as well as the related work of other researchers.

## 2.1 Worm Propagation

As has been mentioned, a worm propagates by sending itself to a remote system and having that system execute its code. The methods for accomplishing this can be split into two broad categories, those which require user intervention and those which do not. Chat programs, e-mail, and file-sharing programs such as Kazaa and Gnucleus are all examples of propagation medium which traditionally require user interaction.

For file-sharing programs and e-mail, a human must download or view the file in order for the code to be executed on the system. Chat programs are similar, except that they tend to provide links to infected web pages, which cause code to be executed when viewed. The alternative method, not requiring user intervention, is for a worm to take advantage of a technical weakness or vulnerability present in a network protocol or a system itself, including the operating system. As they do not depend on the timeliness of human response, worms effecting this method are capable of spreading very quickly. However, this method also tends to be more difficult for worm authors. It requires intimate knowledge of how a vulnerability can be used to remotely execute code and of how to program an exploit for the vulnerability.

In this thesis, we choose to focus on propagation methods which do *not* require user intervention. These may provide an upper-bound on the propagation speed of a worm. This is because it is not required to wait for user interaction. Also, the modeling of non-interactive worms would be somewhat easier than interactive ones. This is because a model for an interactive worm would require attention to be focused on the sociological aspect of certain details. Examples of sociological aspects would include understanding interconnections of people on-line, in a mathematical sense, and various user-dependent wait times.

One major aspect which requires detail in modeling is how a worm finds the vulnerable hosts in a network. E-mail propagation worms use addresses found on the infected computer, while those that propagate through file-sharing programs use the inherent nature of those programs. Non-interactive worms, those which do not require user interaction such as by opening an email attachment, may choose to search for host names or IP addresses on the infected host, however this is a sub-optimal method. These worms, then, have two choices: either carry a hit-list of IP addresses, or use an algorithm to choose a set of addresses to scan. The hit-list approach is a more efficient method, as each iteration of the worm may divide a portion of its work among its children. However, this also brings with it many more bytes of data, and thus a larger footprint. Algorithm-based scanning leads to a more stealthy, smaller worm. Many different scanning methods may be used, however most fall into one of three categories: Sequential, Random, and Biased. The sequential scanning method selects the potential target host by sequentially scanning the address space of all vulnerable hosts, starting at a pre-selected or otherwise chosen unique address. The random scanning method chooses any one of the available hosts with equal probability, $1/2^{32}$ in the case of IPv4. The biased scanning method chooses a particular subnet or domain with higher probability. In practice, the biased method is a variant of random scanning, with a certain percentage of the scans being sent to addresses in the infected host's local subnet.

All of this scanning and transfer of worm code will naturally create network traffic, which may be detectable as an anomaly. If this traffic can be studied and a pattern formed, then early warning systems could be built. To do this, we would need a pattern, or model, of how this propagation traffic differs from or affects normal traffic. With a well designed and realistic model, we can both detect current worm activity and predict future trends.

Worm propagation has previously been studied through three forms: modeling, simulation, and analysis of live events. Actual worm propagation is often difficult to analyze, as it requires a near-to-complete view of the Internet during the time of propagation and may suffer from interference of other worms [SPW02]. Simulation suffers from the need to have a basic model with which to work, although does not require that model to be of a closed form. However, given the expansive nature of the Web and its associated complexity, accurate simulation is difficult. Recently,

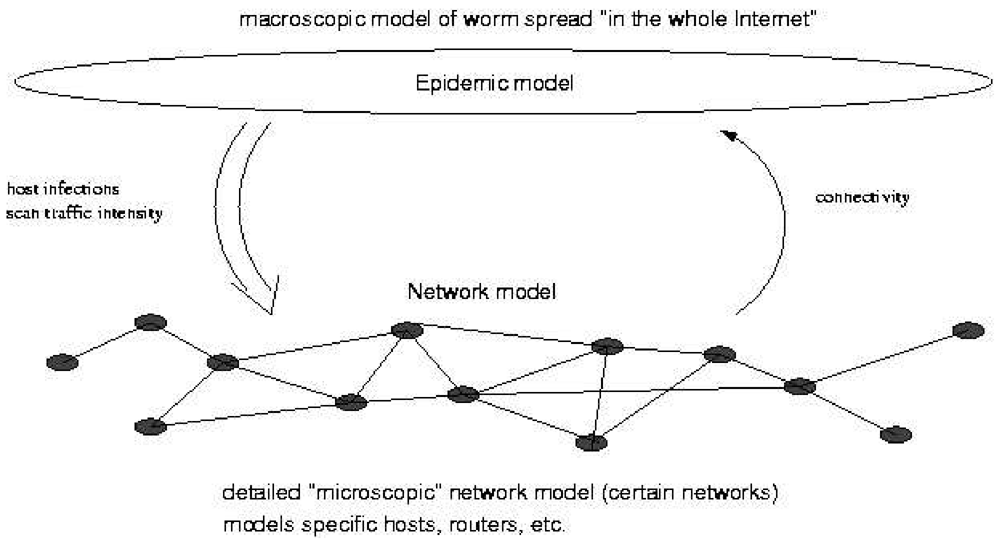
macroscopic model of worm spread "in the whole Internet"
Epidemic model
host infections
scan traffic intensity
connectivity
Network model
detailed "microscopic" network model (certain networks)
models specific hosts, routers, etc.

Figure 1: Abstraction levels from Liljenstam *et. al.* [LNBG03]

several authors [LNBG03] have introduced mixed-mode simulations as an effective tool to understand worm and virus propagations. As shown in Figure 1, particular subnets are simulated in detail (at the packet level), while the Internet as a whole is simulated only from its aggregate behaviors. Specifically, *hot-spot subnets*, where viruses are present and propagating at a rapid rate, are simulated in detail.

For these reasons, most researchers [SPW02, ZTGC03, ZTG03, Vog03] in this area start by creating models with different levels of complexity before attempting simulation. This is then followed with an iterative process of: (1) validating the models through simulation, and (2) working in realistic scenarios that more accurately reflect the true nature of the Internet.

Unfortunately, both the models and simulations used in the past have had a number of weaknesses. Specifically, they do not take into account several important real-world parameters, such as network topology and bandwidth. This thesis remedies these limitations by first using the well-known epidemic model in [SPW02] as the basis for our research, while incorporating the real-world complexities of bandwidth limitations and taking into account the heterogeneous nature of the Internet. That is, not all nodes are equally likely to be infected given a particular virus. Also, not all nodes run the same operating environment and set of applications. We then make changes to a chosen simulator for incorporating our model and a more realistic worm algorithm.

## 2.2 Related Work

In recent years several attempts have been made to model the propagation behavior of worms and viruses. The basic approach, as introduced by Staniford and Paxson *et. al.* [SPW02], describes the propagation behavior in terms of a simple epidemic model. Equations 1 and 2 describe the change in number of susceptible and infected hosts, respectively, over time. In these equations, $\beta$ is the infection rate for a single worm.

$$\frac{ds(t)}{dt} = -\beta * s(t) * i(t) \qquad (1)$$

$$\frac{di(t)}{dt} = \beta * s(t) * i(t) \qquad (2)$$

In order to arrive at this model, Staniford and Paxson made the following assumptions about topology and lifetimes:

- All hosts are equally likely to infect any of the other hosts that have not been infected thus far;
- Infection rate model ignores the presence of firewalls, network address translation (NAT), and other forms of trust or protection; and

- No countermeasures are taken, thus once a system is infected, it remains infected.

The Spatial Epidemic model [LNBG03] attempts to address the topology and countermeasures problems with the Simple Epidemic model. Spatial Epidemic follows the *susceptible-infected-removed* (SIR) pattern, whereby a host is in one of those three states and transitions from susceptible to infected and from infected to removed, as shown here in Equations 3 and 4:

$$\frac{di(t)}{dt} = \beta * s(t) * i(t) - \gamma * i(t) \tag{3}$$

$$\frac{dr(t)}{dt} = \gamma * i(t) \tag{4}$$

In Equations 3 and 4, $\gamma$ represents the removal rate of worms as caused by patching or quarantining. Here, $r(t)$ represents the population so far removed, patched, or quarantined. It also takes into account groupings by modeling different infection rates between and within groups. In the traditional biological sense, this is analogous to different infection rates within and between countries or provinces. In a network sense, this is a simplistic partitioning into subnets or domains.

Subsequently, Zou *et. al.* [ZTGC03] studied the standard epidemic model with random scanning. In their work, they showed that the simple epidemic model holds for a population of uniformly distributed hosts and a worm that applies a divide-and-conquer scanning algorithm, where a worm splits its IP scanning space among its children. In addition, Zou *et. al.* showed that for all considerations of uniformly distributed hosts, the models converge to the standard epidemic model. Finally, in their research they considered a local-preference scanning technique with the assumption that vulnerable hosts are not uniformly distributed. In this latter case, no closed analytical solution was possible. Nevertheless, using the "law of large numbers", they rationalize that averages over the entire population will lead back to the original epidemic model.

Tom Vogt [Vog03] focused on simulation based on the original epidemic model, however provided suggestions for a more expanded study. In his paper, he suggests that network structure and topology has an impact on real propagations. He also calculates the bandwidth consumed by a particular worm spreading in his models. However, he does not consider this bandwidth calculation as a constraint in his model or simulations.

## 3 Approach

The stated hypothesis of this thesis is that real-world constraints, such as bandwidth and heterogeneity of hosts, greatly affect the propagation of worms. This work first attempted to address the claim by making changes to the mathematical models used in previous research. After problems were encountered, a new direction was taken, focusing on creating a generalized algorithmic worm model. This model was then coded into a network simulator [ssf04] for testing and analysis.

As mentioned, the first attempt to address the above claim was to try and work bandwidth and topology considerations into the classic epidemic model[1] used by other researchers [SPW02, ZTGC03, ZTG03, Vog03].

In the Simple Epidemic model, it is assumed that all hosts are universally connected, and thus the probability of any infected host contacting and infecting another host is:

$$p = 1 - (1 - \Omega)^{\eta * \delta} = \frac{\eta * \delta}{\Omega} \qquad ; for \;\; \Omega >> 1 \tag{5}$$

[1]The SIR (Susceptible-Infected-Removed) model as used in biological infection studies.

where $\eta$ is the number of scans an infected host sends per unit time, $\eta * \delta$ is the number of scans an infected host will send in time $\delta$, and $\Omega$ is the total number of hosts including susceptible and non-susceptible. This follows from:

$$p_i = P(\textit{not contacting live host in time slot } i) = 1 - \frac{1}{\Omega} \tag{6}$$

and the product of time slots 0 through $\eta * \delta$, as is evident in Figure 2.

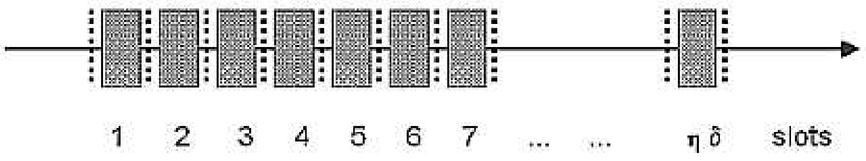

Figure 2: Time slots 0 through $\eta\delta$, where one scan occurs in one time slot.

We proposed an extension to the Simple Epidemic model that adjusts for bandwidth considerations and non-uniform infection probabilities. Whereby:

$$\beta_{ij} = \frac{\eta_{ij}}{\Omega} \tag{7}$$

and the simple probability, $p$, is refined for group-to-group and host-to-host interactions, in the form of $p_{ij}$. We made note that although it is ideal to find a closed-form solution for these equations, it is not necessary to gain necessary insights through simulation.

To adequately test such changes to the model, real-world data of bandwidth constraints and hosts' software needed to be gathered; specifically, in support of these questions:

1. What is the range and average up-link bandwidth for hosts on the Internet?

2. What is the percentage and connectivity of hosts on the Internet that have

certain operating systems and server software installed?

Network research mailing-lists, individuals, and companies were contacted in seeking this information, but with little results. Responses from individuals and the mailing lists included no direct knowledge of this information, however provided pointers to organizations such as IANA, ICANN, NLANR, and CAIDA. Of these, only NLANR[2] and CAIDA[3] seemed to be collecting network information, but did not provide the necessary bandwidth statistics. Telegeography[4], a company which creates and sells reports of Internet statistics, responded to our inquiries with a statement that they do not collect inter-network link bandwidths and do not know of anyone else who does. For operating system usage, The Gartner Group and the Network Operations of WPI were contacted. The Gartner Group had statistics for how many PCs are sold with different operating systems on them, but not on actual usage. WPI Network Operations reported that they do not collect such information. While there are a few particular instances of some operating system and software usage statistics on-line[5], often their breadth, and sometimes accuracy, are lacking.

These problems, coupled with the mathematical difficulties of forming a complete and closed model, led to a reassessment of the claim put forth and approach to be taken toward it. During this reassessment, a few realizations were made concerning the state of current worm research. First, it became obvious that most, if not all, of the research on worm propagation had been concerning the effects of worms on the Internet as a whole and not on particular segments of subnets. Secondly, there is a lack of tools available to test new worm actions and propagation routines without releasing them to the wild or in a completely controlled, small-scale, set-

[2]NLANR's AMP and PMA projects

[3]CAIDA's Skitter project

[4]http://telegeography.com/

[5]Many websites provided statistics of their users, including operating system and browser used. Netcraft, http://www.netcraft.com/, provides statistics of web server proliferation.

ting. To address both of these issues in the scope of the original hypothesis, a new approach was chosen that considered creating a parameterized, *algorithmic* worm model and a worm simulation tool capable of simulating a campus-sized network and its interactions with the Internet as a whole.

## 4 Worm Design

In order to build a realistic simulator for any type of worm imaginable, a component-based architecture was developed to be easily varied and extended to form all sorts of worm instances. In this section, we described the proposed architecture, its components, and the set of parameters being used to simulate a worm instance within a local corporate network environment.

By definition, a worm is a self-contained, self propagating program. Thus, in simple terms, it has two main functions: that which propagates and that which does "other" things. We propose that there is a third broad functionality of a worm, that of self-preservation. We also propose that the "other" functionality of a worm may be more appropriately categorized as Goal-Based Actions (GBA), as whatever functionality included in a worm will naturally be dependent on whatever goals (and subgoals) the author has.

The work presented by Weaver *et. al.* in [WPSC03] provides us with mainly an action and technique based taxonomy of computer worms, which we utilize and further extend here.

### 4.1 Propagation

The propagation function itself may be broken down into three actions: acquire target, send scan, and infect target. Acquiring the target simply means picking a host to attack next. Sending a scan involves checking to see if that host is receptive to an infection attempt, since IP-space is sparsely populated. This may involve a simple ping to check if the host is alive or a full out vulnerability assessment. Infecting the target is the actual method used to send the worm code to the new host. In algorithm form:

```
propagate() {
  host := acquire_target()
  success := send_scan(host)
  if( success ) then
    infect(host)
  endif
}
```

In the case of a simple worm which does not first check to see if the host is available or susceptible (such as Slammer), the scan method is dropped:

```
propagate() {
  host := acquire_target()
  infect(host)
}
```

Each of these actions may have an associated ”cost” to its inclusion and execution, such as increased worm size and CPU or network load. Depending on the authors needs or requirements, these become limiting factors in what may be included in the worm's actions. This is discussed further after expanding upon these actions below.

### 4.1.1 Target Acquisition

The Target Acquisition phase of our worm algorithm is built directly off of the *Target Discovery* section in [WPSC03]. Weaver *et. al.* taxonomize this task into 5 separate categories. Here, we further extend their work through parameterization.

**Scanning** Scanning may be considered an equation-based method for choosing a host. Any type of equation may be used to arrive at an IP address, but there are three main types seen thus far: sequential, random, and local preference. Sequential scanning is exactly as it sounds: start at an IP address and increment through all the IP space. This could carry with it the options of which IP to start with (user chosen value, random, or based on IP of infected host) and how many times to increment (continuous, chosen value, or subnet-based). Random scanning is completely at random (depending on the chosen PRNG method and its seed value). Local preference scanning is a variance of either Sequential or Random, whereby it has a greater probability of choosing a local IP address over a remote one (for example, the traditional 80/20 split).

**Pregenerated Target Lists** Pregenerated Target Lists, or so called "hit-lists", could include the options for percentage of total population and percentage wrong, or just number of IPs to include. Implicit to this type is the fact that the list is divided among a parent and its children, avoiding the problem of every instance hitting the exact same machines.

**Externally Generated Target Lists** Externally generated target lists depend on one or more external sources that can be queried for host data. This will involve either servers that are normally publicly available, such as gaming meta-servers, or ones explicitly setup by the worm or worm author. The normally available meta-servers could have parameters for rates of change, such as many popping up at night or leaving in the morning. Each server could also have a maximum queries/second that it would be able to handle. The worm would also need a way of finding these servers, either hard-coded or through scanning.

**Internal Target Lists** Internal Target Lists are highly dependent on the infected host. This method could parameterize the choice of how much info is on the host, such as "all machines in subnet", "all windows boxes in subnet", particular servers, number of internal/external, or some combination.

**Passive** Passive methods are determined by "normal" interactions between hosts. Parameters may include a rate of interaction with particular machines, internal/external rate of interaction, or subnet-based rate of interaction.

Any of these methods may also be combined to produce different types of target acquisition strategies. For example, the a worm may begin with an initial hit-list of 100 different hosts or subnets. Once it has exhausted its search using the hit-list, it may then proceed to perform random scanning with a 50% local bias.

It is important to note, however, that the resource consumption of each method is not the same. Different methods may require the worm to be large, such as the extra bytes required by a hit-list, or to take more processing time, such as by searching the host for addresses of other vulnerable hosts. Further research and analysis should be performed in this area to determine associated costs for using each method. The costs could then be used in determining design trade-offs that worm authors engage at. For example, hit lists provide a hight rate of infection, but at a high cost of worm payload size.

### 4.1.2 Sending a Scan

The send_scan function tests to see if the host is available for infection. This can be as simple as checking if the host is up on the network or as complex as checking if the host is vulnerable to the exploit which will be used. The sending of a scan before attempted infection can reduce the scanning rate if the cost for failing an infection is greater than the cost of failing a scan or sending a scan plus infection. One important parameter to this would be the choice of transport protocol (TCP/UDP) or just simply the time for one successful scan and time for one failed scan. Also, whether or not it tests for the host to be up or if it is a full test for the vulnerability (or for multiple vulnerabilities).

### 4.1.3 Infection Vector (IV)

The particular infection vector used to access the remote host is mainly dependent on the particular vulnerability chosen to exploit. In a non-specific sense, it is dependent on the transport protocol chosen to use and the message size to be sent. Section 3 of [WPSC03] also proposes three particular classes of IV: Self-carried, second channel, and embedded.

## 4.2 Self Preservation

The Self Preservation actions of a worm may take many forms. In the wild, worms have been observed to disable anti-virus software or prevent sending itself to certain antivirus-known addresses. They have also been seen to attempt disabling of other worms which may be contending for the same system. We also believe that a time-based throttled scanning may help the worm to "slip under the radar". We also propose a decoy method, whereby a worm will release a few children that "cause a lot of noise" so that the parent is not noticed. It has also been proposed [WPSC03] that a worm cause damage to its host if, and only if, it is "disturbed" in some way. This module could contain parameters for: probability of success in disabling anti-virus or other software updates, probability of being noticed and thus removed, or "hardening" of the host against other worms.

### 4.3 Goal-Based Actions

A worm's GBA functionality depends on the author's goal list. The *Payloads* section of [WPSC03] provides some useful suggestions for such a module. The opening of a back-door can make the host susceptible to more attacks. This would involve a probability of the back-door being used and any associated traffic utilization. It could also provide a list of other worms this host is now susceptible to or a list of vulnerabilities this host now has. Spam relays and HTTP-Proxies of course have an associated bandwidth consumption or traffic pattern. Internet DoS attacks would have a set time of activation, a target, and a traffic pattern. Data damage would have an associated probability that the host dies because of the damage.

## 5 Implementation

This section describes the implementation of our algorithmic model defined in Section 4. Requirements for choosing a simulator are presented, along with the design of the framework.

In order to test our hypothesis, that bandwidth and heterogeneity of hosts affect worm propagation, a network-based simulator was sought that would allow effective and easy incorporation of our algorithmic model. The reason for choosing a network-based simulator is because such a tool allows the level of detail, specifically packet-level, to show worm traffic effects with network-based constraints. Different simulator options were considered before choosing the one that best fit the needs of this research.

The ns2 network simulator [ns2] was considered because of prior experiences and general community knowledge of it. ns2 is a fully packet-level network simulator written in C++ and OTcl. It is both fast, for a packet-level simulator, and very scriptable. It provides constructs for simple hosts and routers that have different networking protocols for sending send packets between each other. It has mainly been used in the past for analyzing changes to routing algorithms, the TCP/IP protocol, and active queue management algorithms.

The Network Worm Simulator (NWS) [nws] was briefly considered, and mentioned here, mainly due to its name. This is a simulator written in Perl from the ground-up. It is based on the idea of message-passing between hosts, however is not on the level of network packets or any form of IP stack. For this reason, NWS was not chosen for this project.

The Scalable Simulation Framework (SSFNet) [ssf04] was utilized within the work of Liljenstam *et. al.* in [LYPN02, LNBG03]. SSFNet is a packet-level simulator, much like ns2, written in Java and utilizes a Domain Modeling Language (DML) specifically developed for writing simulations in SSFNet. Michael Liljenstam has begun creation of a worm framework (SSF.App.Worm [ssf]) within the SSFNet simulator. For this reason, the SSFNet simulator was chosen, specifically to build off of the SSF.App.Worm framework currently in place.

## 5.1 Current state of SSF.App.Worm

While reading through [LYPN02] and [LNBG03], it seemed that the current state of the SSF.App.Worm framework included worm packets being created on the LAN level. However, upon further investigation, it became uncertain that this was truly the case. The creator of the framework, Professor Michael Liljenstam, was contacted for confirmation and further guidance. He provided much clarification and further help, as may be found in Appendix C, including the providing of beta code which was currently under development. Professor Liljenstam stated that this new code (SSF.App.Worm v0.6) provides for packet-level simulation of worm flows. However, it was determined that learning the capabilities of the new code and then extending for our purposes might not provide for timely completion. Thus, it was decided to utilize SSF.App.Worm's iterative capabilities on the macroscopic level and build our own framework for the packet-level and for translating between the two.

As mentioned above, the SSF.App.Worm framework currently takes epidemic model parameters (initial number susceptible, initial number infected, infection rate) and runs a time-stepped iteration over the Epidemic equations. These equations are used to calculate when particular susceptible hosts within LANs become infected, as well as track the total infection population, number of incoming scans to LANs, and number of outgoing scans from LANs. This capability was utilized within our new code, while further extending it and creating both the packet-level LANs and translation between packet and macroscopic levels.

## 5.2 Framework

A framework was constructed that incorporates the algorithmic model of Section 4 into the SSFNet simulator.

Similar to the ideas of [LYPN02], there needed to be an Internet-level module which keeps track of the worm spreading throughout the Internet as a whole. This is on a very high level, being simply equation-based and not packet-based. The abstraction is both necessary, for scalability, and reasonable. It has been shown

[SPW02, ZTG03, LYPN02] that epidemic models can be created to model the spread of worms on such a high level.

The next two key components are on the LAN-level of simulation: Hosts and Gateways. A Gateway is simply the gateway router that provides the up-link from the LAN to the Internet. Hosts are individual hosts within a LAN that are made up of multiple components themselves. Hosts have a particular operating system version installed and are of a single architecture (such as x86 or alpha). They also run particular software which may provide network services, such as web (HTTP) or email (SMTP).

The third key component is the Worm itself, which builds on the algorithmic model of Section 4 and thus described in brief here. The three parts of a Worm are the propagation mechanism, the Goal-Based Actions (GBAs), and self-preservation techniques. The propagation is broken down into targeting and infecting, with an optional scanning in between. Targeting is the method by which the worm chooses an IP address of its next target. Infecting is the sending of the actual infection vector (IV). Scanning may be anything from a simple ping-check of the host to a full vulnerability check. GBAs are actions based on the goals of the worm author, such as installing a spam relay or DoS attack. Self-preservation techniques may involve such things as delaying a worm's infection rate to make it more stealthy or disabling anti-virus software on the infected host.

Many changes needed to be made to the simulator to incorporate this designed framework, the most important of which was the translation between Internet level equations and packet level LANs. Specifics of these changes may be found in appendix B.

# 6 Testing, Results and Analysis

This section describes the testing procedures used, results obtained, and analysis thereof. The hypothesis being tested is that real-world constraints affect the propagation of a worm, particularly within a LAN environment.

## 6.1 Experimental design

All of the following simulations were run multiple (10) times, with the data from each then averaged and shown below. The reason for multiple runs of the same simulation was to produce an *average* behavior for analysis, rather than dealing with statistical aberrations from analyzing a single run. We also wished to have a certain degree of *confidence* to guarantee that our ten runs were sufficient.

A simple confidence metric to use when sampling data is to test if most values fall within two standard deviations ($2*\sigma$) of the mean. To accomplish this, we must compute the sample variance:

$$\sigma^2 = \Sigma(x_i - x_{mean})^2/(n-1) \tag{8}$$

where $\sigma$ is the standard deviation, $x_i$ is the $i^{th}$ sample value, $x_{mean}$ is the mean over all $x_i$'s, and $n$ is the number of samples. We must then compute the averages and confidence factors ($2*\sigma$) for both the times to infections in the LAN and the rate of infection[6]. These averages and confidence factors may be found in Tables 1 and 2. All data values fell within the confidence interval ($mean$ +/- $2*\sigma$), which led us to reason that the ten runs of each simulation were sufficient to analyze their aggregate, average behavior and perform meaningful analysis.

It is important to note that in order to obtain varying simulation results, the

[6]The rate of infection is equivalent to the slope of the graph.

| **% Infected in LAN** | **Average time to reach %** | **Confidence Factor** |
|---|---|---|
| 10 | 85.3890755 | 33.4283144158765 |
| 20 | 98.0671414 | 30.333730906725 |
| 30 | 107.0233175 | 27.3126402818765 |
| 40 | 115.8380135 | 21.2708501069923 |
| 50 | 121.5869975 | 17.1973449459653 |
| 60 | 126.3956217 | 13.3913031857912 |
| 70 | 132.1420465 | 13.3404478406287 |
| 80 | 139.69174 | 12.9847895768754 |
| 90 | 148.60002 | 17.3014541445587 |
| 100 | 171.992899 | 38.6074394331288 |

Table 1: Confidence intervals for time to infection in the LAN.

| **% Infected in LAN** | **Average Rate of Infection** | **Confidence Factor** |
|---|---|---|
| 10-20 | 12.6780659 | 15.698609116824 |
| 20-30 | 8.9561761 | 19.6927089049045 |
| 30-40 | 8.814696 | 14.5981915075843 |
| 40-50 | 5.748984 | 5.86531587335395 |
| 50-60 | 4.8086242 | 7.52210120758779 |
| 60-70 | 5.7464248 | 7.47088397351632 |
| 70-80 | 7.5496935 | 10.7342101089244 |
| 80-90 | 8.90828 | 12.3830007750231 |
| 90-100 | 23.392879 | 44.3957923584191 |

Table 2: Confidence intervals for rate of infection in the LAN.

PRNG seed value specified in the DML configuration file was changed for each run, thus producing different sequences of random numbers.

The worm chosen to simulate was SQL Slammer [MPS+03]. This worm was of interest to us for two reasons. It propagated so quickly and caused so much network traffic that it utilized the entire up-link bandwidth of many LANs. Also, Slammer has only a propagation method, no GBAs. Also, that propagation method consisted solely of choosing a target IP address and sending a single, 376 byte UDP packet to that target. This simplicity made coding of the worm in the simulator much easier. The framework, however, provides for more complicated worms to be encoded, as is evident by our adjusting the original Slammer propagation function to use a locally-biased target acquisition method.

For all simulations involving the new code, a LAN topology was created based on that of the Worcester Polytechnic Institute network. This involved approximately 1700 hosts separated into various subnets by routers. All subnets were either directly or indirectly connected to a backbone network of six routers with gigabit links connecting them in a ring topology, with one of the routers providing an up-link to the *Internet*. Further details of this topology, as well as the DML configuration file, may be found in Appendix A.

## 6.2 Establishing a baseline

The first step to testing a new claim is to create a baseline with which to compare results. This often involves reproducing prior research tests and then running variations on those tests. Since this work both built heavily off of the SSF.App.Worm simulations and introduces a new extension, the first step was to run the chosen worm, Slammer, with the original SSF.App.Worm code and then with the newly introduced code.

Since the original SSF.App.Worm code iterates over the epidemic model equations, it required the epidemic parameters take on those values associated with the Slammer worm [ZGT03], as specified in Table 3. $s\_0$ is the number of initially susceptible hosts in the Internet, $i\_0$ the number of initially infected hosts in the Internet, and $\beta$ the infection rate. The LAN topology used is based on the WPI campus network and more fully explained in Appendix A. The results from this simulation are shown in Figure 3 as the *original code* data line. The first infection within the LAN occurs at 204 seconds, with the last at 249 seconds.

| **Parameter** | **Value** |
|---|---|
| $s\_0$ | 75000 |
| $i\_0$ | 10 |
| $\beta$ | $9.313e-7$ |

Table 3: Slammer epidemic parameter settings.

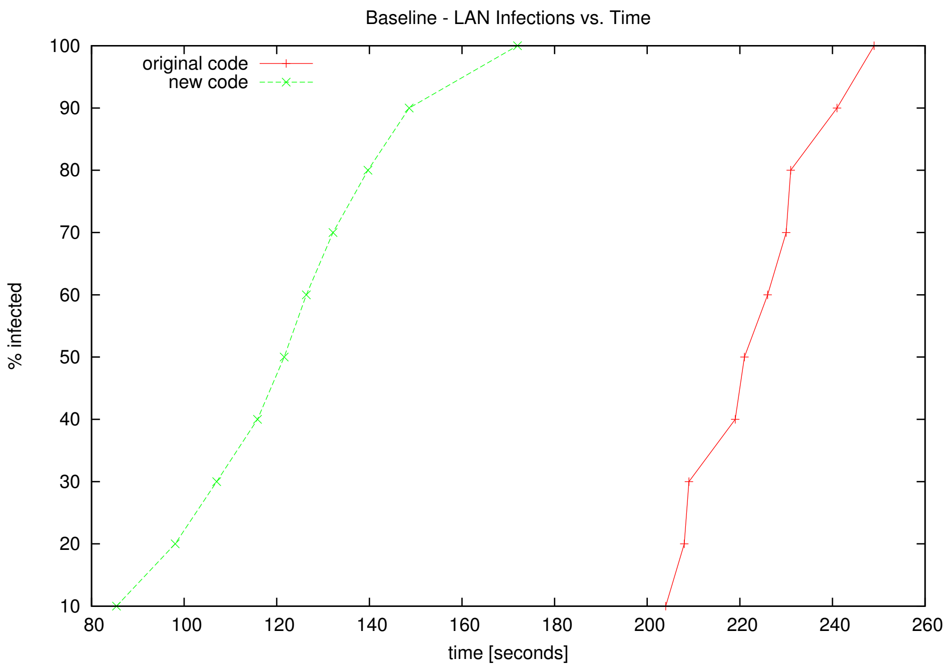

Figure 3: Creating a baseline - LAN infections versus time

The new SSF.App.Worm code, which includes the worm framework as described in Section 5, both utilizes the original iterative mechanism and introduces new parameters. The original mechanism is used for the global infection rate, while the new parameters are specific to the LAN being simulated at the packet level. Important parameters introduced are: up-link capacity, number in LAN susceptible, number in LAN initially infected, and scanning strategy. In creating a base-case, these parameters for the LAN were set as detailed in the first row of Table 4. The results from this simulation are shown in Figure 3 as the *new code* data line. On average, the first infection within the LAN occurs at $t = 85.4$, with the last at $t = 172.0$

Two observations may be made from the data in Figure 3. First, once the initial infection occurs within the LAN, the rate of propagation between original code and new code is similar. Second, the first infection within the LAN occurs sooner with the new code than with the original code. We believe this difference of initial infection time to be due to the differing methods in which the new code and the original code calculate when a susceptible host becomes infected.

The original code infects the LAN susceptible hosts using solely the mathematical epidemic equations. For each time step, it calculates how many new global infections there were since the last time step. Then, with probability based on the number of LAN susceptible hosts versus the number of globally susceptible hosts, one of the LAN susceptible hosts is chosen to become infected. Mathematically, this is equivalent to:

$$P\_infect_{original} = (i_{ttl} - i_{old}) * \frac{s_{net}}{s_{global}} \tag{9}$$

where $i_{ttl}$ is the total number of global infections at time $t$, $i_{old}$ is the total number of global infections at time $t-1$, $s_{net}$ is the number of susceptible in the LAN at time $t$, and $s_{global}$ is the number of susceptible in the Internet at time $t$.

Our new code infects the LAN susceptible hosts using a combination of the epidemic equations and sending of simulated scan packets on the network. We assume that the number of scans created at a time, $t$, is based on the number of infected hosts at that time. The probability that a LAN susceptible host becomes infected is based on the total number of new scans created globally, the probability that those scans are sent to the LAN, and the probability that those scans sent to the LAN will reach a susceptible host. Mathematically, this is equivalent to:

$$P\_infect_{new} = i_{ttl} * 2^{32} * \beta * \frac{lan\_space}{2^{32}} * \frac{s_{net}}{lan\_space} \tag{10}$$

where $i_{ttl}$ and $s_{net}$ are as in Equation 9, $2^{32}$ is the total IP space for IPv4, $\beta$ is the infection rate as described in the standard epidemic model, and $lan\_space$ is the total IP space for the LAN.

To simplify and compare Equations 9 and 10, we substitute an equivalent equation for $i_{ttl}$,

$$\begin{aligned} \frac{di}{dt} &= \beta * s_{global} * i_{global} && (11) \\ i_{ttl} &= i_{old} + di && (12) \end{aligned}$$

and arrive at

$$\begin{aligned} P\_infect_{original} &= \beta * s_{global} * i_{old} * \frac{s_{net}}{s_{global}} && (13) \\ &= \beta * i_{old} * s_{net} && (14) \\ P\_infect_{new} &= i_{ttl} * \beta * s_{net} && (15) \\ &= (i_{old} + \beta * s_{global} * i_{old}) * \beta * s_{net} && (16) \\ &= i_{old} * \beta * s_{net} + \beta^2 * s_{global} * s_{net} * i_{old} && (17) \end{aligned}$$

As may be seen, the probability of infecting a susceptible host within the LAN is slightly greater for the new code. It should be understood that the calculation of the number of scans to send causes those scans to be created at the network level and sent across the 30ms link to the LAN. This small up-link delay causes a host to become infected almost instantaneously when its IP address has been selected for a scan packet. We believe that this difference between Equations 9 and 10 may be due to the time step involved, and would be further tested by decreasing the resolution of the time step from one second to a value on scale with the LAN's up-link delay. We note, however, that it is apparent in Figure 3 that the *rate* of infection is not affected by this discrepancy, and thus we may gain insights through analysis of test results with the new code.

## 6.3 Parameter variations

| **ID** | **Up-link capacity** | **LAN Susceptible** | **Scan Strategy** | **Feedback?** |
|---|---|---|---|---|
| 1 | 45Mb | 10 | random | No |
| 2 | 45Mb | 20 | random | No |
| 3 | 45Mb | 5 | random | No |
| 4 | 15Mb | 10 | random | No |
| 5 | 15Mb | 10 | random | Yes |
| 6 | 5Mb | 10 | random | Yes |
| 7 | 45Mb | 20 | 50%, 8-bit local bias | No |
| 8 | 45Mb | 20 | 50%, 16-bit local bias | No |
| 9 | 45Mb | 20 | 50%, 8-bit local bias | Yes |

Table 4: Simulation parameter settings

Table 4 describes the main parameter variations between each simulation run in this section. The **ID** is simply an identifier for that run, such as referring to $ID = 1$ as the first simulation and $ID = 4$ as the fourth simulation. **Up-link capacity** is the capacity of the LAN's sole link to the Internet, across which all

Internet-to-LAN and LAN-to-Internet scans must traverse. **LAN Susceptible** is the number of susceptible hosts in the LAN. **Scan Strategy** is the target acquisition method used by the worm. **Feedback** is whether or not the simulation incorporates a feedback mechanism, further described below. All simulations were run with no initial infections in the LAN, thus the first LAN infection occurs due to an incoming Internet-to-LAN scan, as is normally the case but for the LAN where the worm is released.

We believed that each of these parameters would show effects as predicted by our hypothesis. In particular, we reasoned that the up-link capacity for a LAN should throttle the propagation of a worm within that LAN. Differing the number of susceptible hosts in the LAN allowed us to test the hypothesis that the number of hosts affects the observed rate of infection. Differing the scanning strategy was also believed to show how a worm's simple epidemic parameters were not adequate to describe localized, corporate LAN scale, propagation. The feedback mechanism was worked into the mix in order to further show how bandwidth constraints can affect not only the localized propagation, but the global as well.

The first set of variations were aimed at testing a change in the number of susceptible in the LAN. These simulations were equivalent to an increase of twice as many susceptible and a decrease to half as many susceptible. Figure 4 shows the number of infections in the LAN over time, comparing these simulations with the base-case ($ID = 1$). As can be seen, the initial infection occurs sooner for the simulation with 20 susceptible and later for the simulation with 5 susceptible. This follows logically, as the probability that one host will be chosen in a single scan is one over the total IP space, or $1/2^{32}$. Thus with 20, 10, and 5 susceptible, the chance that one of the susceptible will be chosen to infect is $20/2^{32}$, $10/2^{32}$, and $5/2^{32}$, respectively. Similar reasoning may be made for the tail of the curves.

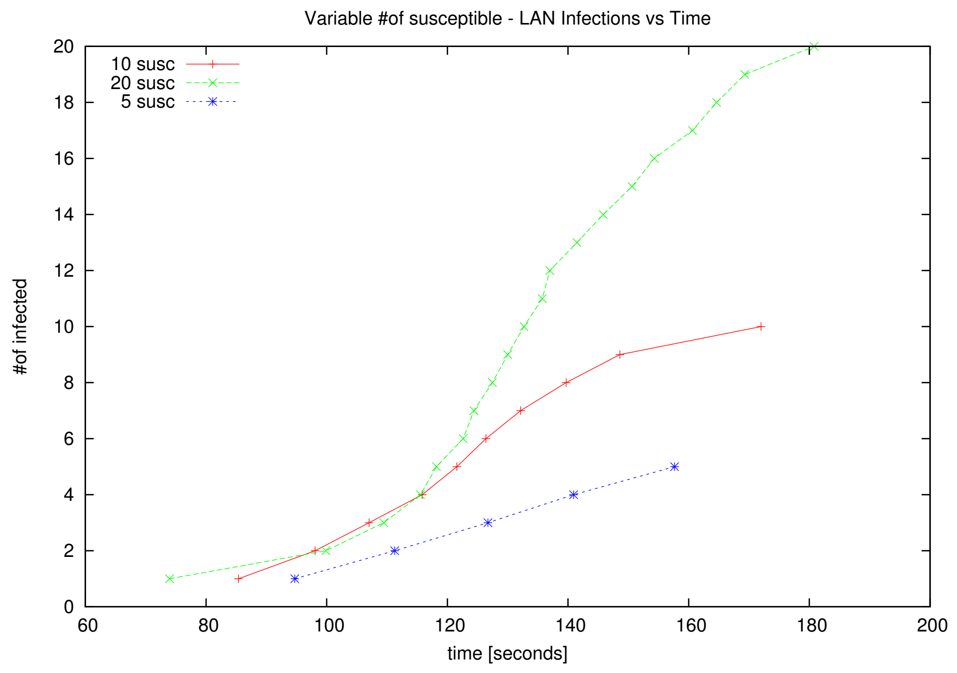

Figure 4: Varying number of susceptible - Infections versus time

The number infected at $t = 150$ is approximately half the base-case with half the number of susceptible, and approximately twice the base-case with twice the number of susceptible. While we admit that these results are entirely expected, they help to verify appropriate behavior of the simulator.

The next set of variations, simulations four through six, were run to show the effects of a throttled bandwidth on the worm's scan traffic and infection rate. Figure 5 shows the outgoing scan rate from the LAN for each of these simulations, plus the base-line. The important thing to note here is that the steady-state reached by the outgoing scans should be equivalent in all cases, as the total number of worm infections is the same. However, as can be seen, the baseline (with a 45Mbps up-link) has approximately a three-times greater steady state than the 15Mbps case, and that the 15Mbps case has approximately a three-times greater steady state than the 5Mbps case. This shows that there is truly throttling of outgoing scans due to

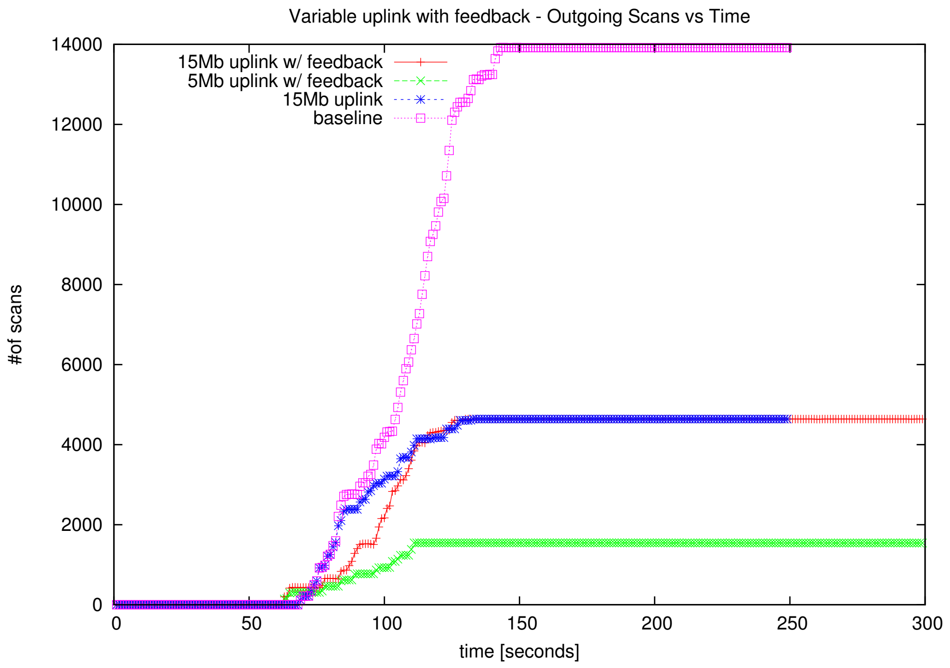

Figure 5: Varying up-link capacity - LAN's outgoing scan rate

the up-link capacity. Why this is important will become clear shortly.

We now introduce the concept of a feedback mechanism into the simulations. By feedback, we mean that the outgoing scan-rate from the LAN should affect the global propagation rate. Specifically, the proportion of scans successfully traversing the LAN's up-link line, versus those attempting to traverse the LAN's up-link line, should be proportional to a change in the global infection rate, $\beta$. This mechanism was developed and encoded into the simulator in part to test the claim that bandwidth constraints should be considered when constructing worm propagation models. A major assumption of this feedback method is that the proportion of successfully outgoing scans is the same across all LANs. While this may not provide *exactly* correct proportions, it is a reasonable abstraction to make in order to show that there is indeed necessity to create worm propagation models with bandwidth constraint considerations.

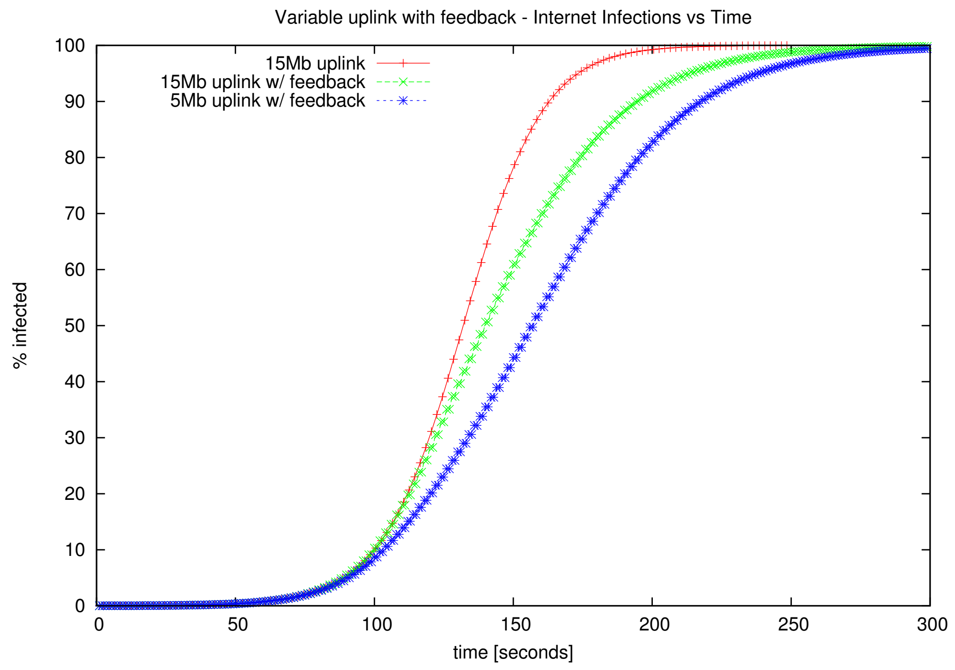

Figure 6: Varying up-link capacity - Internet infection rate

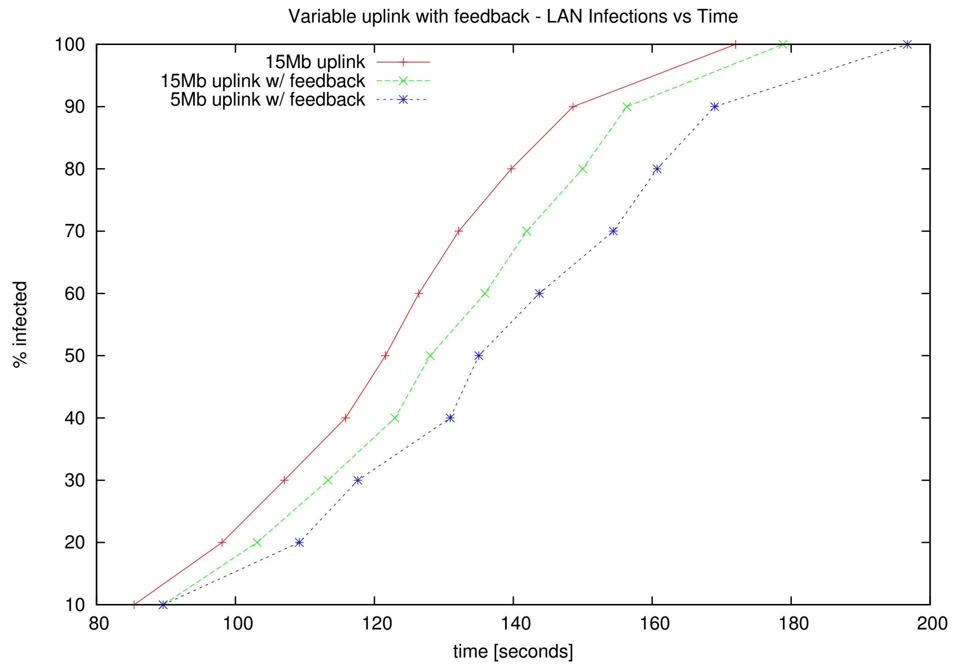

Figure 7: Varying up-link capacity - LAN infection rate

As we hypothesized, the global infection rate and LAN infection rate do indeed change when the feedback mechanism is introduced, shown in Figures 6 and 7, respectively. As may be seen for both the Internet and the LAN, infection rate is effectively slowed, so that 100% infection takes more time.

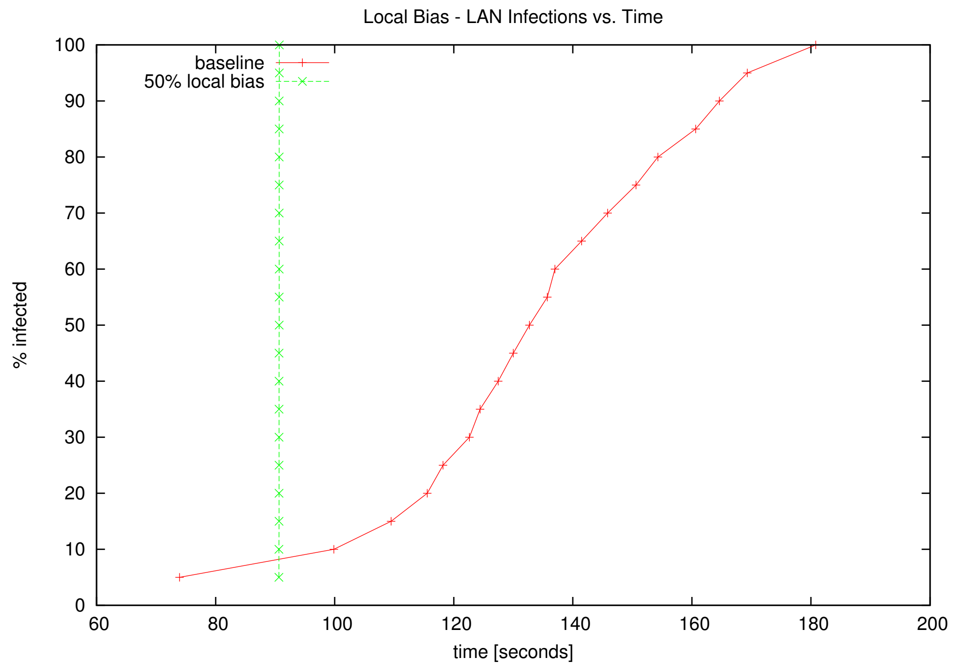

Figure 8: Vary in scanning strategy - 50% local bias

The final set of simulations were concerned with varying the scanning strategy of the worm. While Slammer itself used only a random target acquisition method, we thought it appropriate to see how a change to a locally-biased strategy would affect the propagation of the worm.

The first simulation run was that of a simple 50%, 8-bit local bias, with parameters set according to row six of Table 4. As can be seen in Figure 8, as soon as the first host becomes infected within the LAN, all of the other susceptibles are nearly immediately infected. This is because half of the scans for that infection are now being sent to the 255 hosts within its class C address space, which all of the other

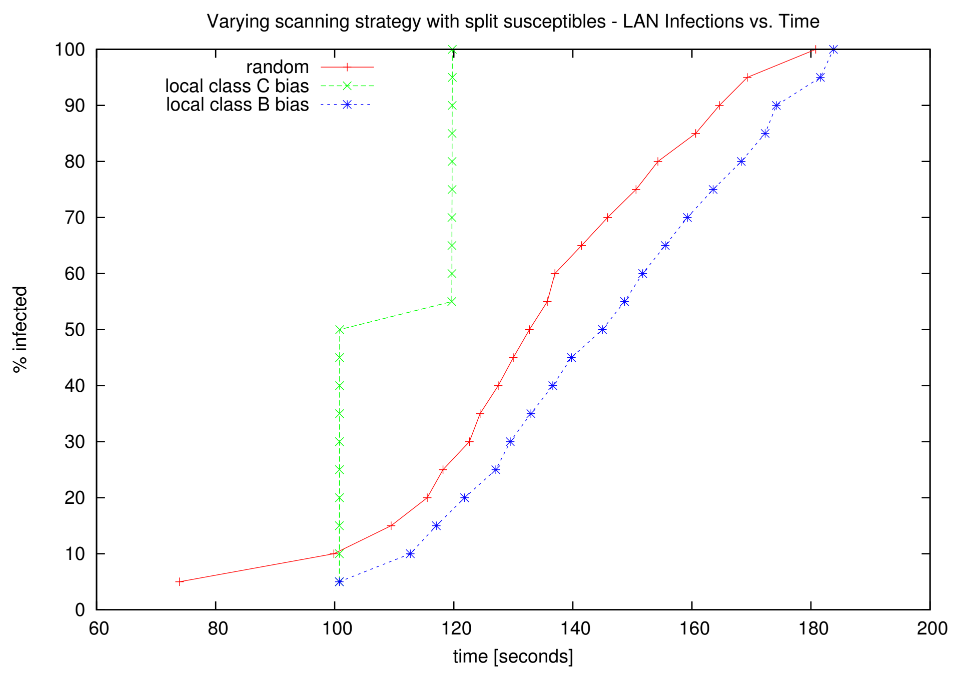

Figure 9: Vary in scanning strategy - split susceptible

susceptibles are in.

After seeing these results, we were curious to know what would happen when the susceptible population is broken up, as is the case in a realistic LAN topology. We then separated ten susceptible hosts from the initial class C subnet, placed them in a different subnet within the LAN, and re-ran the simulations. Since the susceptible hosts were now spread across the campus, we also felt it appropriate to see what would happen when we changed the idea of locality from class C address space to a class B address space (a 16-bit mask on the infected host's IP address). Results from these simulation are displayed in Figure 9.

We can see that the initial infections in the LAN occur similarly to that in Figure 8; as soon as the first host is infected, all susceptible hosts in its proximity are immediately infected. However, it takes some time before the next set of hosts become infected. While still allowing the worm to propagate faster in the LAN, we believe this disconnect provides an opportunity for mitigation. There will naturally be an unusual traffic pattern produced by the worm, which may be noticed within the initial subnet and cause blocking of the worm or that service across the entire LAN, thus preventing the remaining susceptible hosts from becoming infected. This may particularly be useful when utilizing honeynets [hon04].

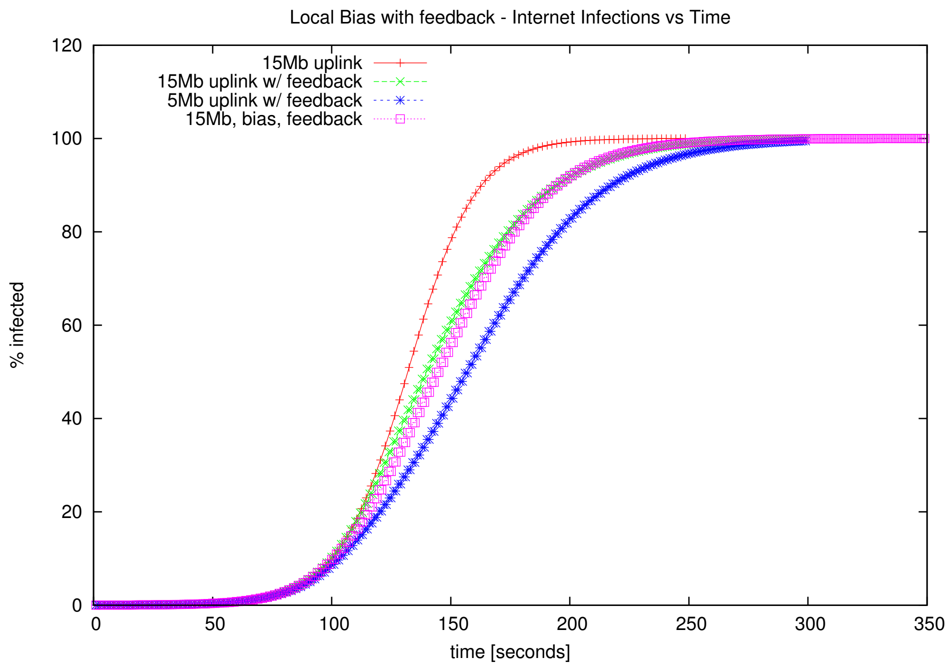

Figure 10: Vary in scanning strategy - Local bias with feedback

Another question arose from these local-bias simulations, that of how the proposed feedback mechanism would affect the propagation. After enabling the feedback mechanism and re-running the simulations, Figure 10 was formed. In this graph, we see the global infection of the worm. There are four plots: (a) a 15Mb up-link capacity and no feedback mechanism, (b) a 15Mb up-link capacity with feedback, (c) a 15Mb up-link capacity with feedback and a locally-biased worm, and (d) 5Mb up-link capacity with feedback. All plots follow the traditional S-curve, as defined by the epidemic model. However, variations are seen in between when there is a small percentage of infected hosts and when the maximum is reached.

The similarities between the plots, for the initial and maximal infection, may be easily explained. The feedback mechanism, which effectively only adjusts $\beta$ in the epidemic model, does not take effect until there is throttling of outgoing scans in the local LAN. Since the first infection occurs around $t = 100$, then the feedback will not start to affect the infection rate until this time, as seen in the graph. Each plot then eventually reaches the 100% mark and stays there. As there are no removals occuring in the system, this is the natural steady-state that all infections are expected to reach, as seen in the graph. As expected, the feedback mechanism causes the global infection rate to slow down from plot (a) to (b). There is a slight slow-down even further, from plot (b) to (c). Since the change is to a locally-biased worm, it can be assumed that the quicker local infection rate, as shown in Figure 8, causes the up-link to be saturated sooner. This then causes the feedback mechanism to take effect sooner, thus showing a slightly slower propagation rate. When the up-link capacity is shrunk even further, to 5Mbps in plot (d), the up-link is saturated even sooner, thus causing the slower infection rate.

## 7 Conclusions and Future Work

This thesis set out to test the claim that real-world parameters, such as up-link capacity, topology, and worm scanning strategy, do affect both the propagation of worms within a LAN and their effects on that LAN. Problems were identified for the worm research community, with respect to lack of data and appropriate tools. An algorithmic worm model was developed, specifically proposing the concept of Goal-Based Actions (GBAs) that affect the combination of various worm functions. A simulation framework was developed[7] to allow for testing of different parameters of the worm algorithm. This simulation framework was then used to test the proposed hypothesis. Variations of up-link bandwidth and scanning strategy showed that there is an impact on the local network and the rate of outgoing scans. A feedback mechanism was introduced to allow changes in the LAN outgoing scan rate to affect the global infection rate. This feedback showed that both the global and local propagation are affected by bandwidth constraints. We believe these results present reasonable justification for the hypothesis.

Future work will include further testing to verify the results obtained in Section 6. In particular, the results shown in Figure 9 should be verified by running more variations on the parameters of locality (number of bits in the scanning strategy net-mask), percentage of bias, and LAN topology. The results from these variations may then be used to gain further insight into how a worm's locally-biased target acquisition strategy propagates differently or similarly on different LANs. The concept of *feedback* should also be further studied. While in this work it was adequate to approximate the throttling imposed by a LAN's up-link capacity, further analysis and thoughtful inspection should lead to possible changes in the epidemic model equations. For instance, we could make a change in the global $\beta$ based on an assumed global bandwidth limitation. This global bandwidth limitation would require either research and simulations of various different LAN topologies to gain insight of the average throttling of outgoing scans. While the strategy used in this thesis[8] was adequate to show that a feedback behavior exists, it is flawed because the simulation was of a single type of LAN configuration.

The next step would naturally involve simulating more types of worms, especially

[7]The framework will be open available for researchers through the WPI System Security Research Lab, http://wssrl.wpi.edu/

[8]That of calculating the percentage of successfully sent scans at the up-link router.

TCP-based ones. Data should be gathered to effectively form methods for properly encoding the worms into the simulator. Of particular note will be the method of translating between global- and packet-level network traffic. Data from past worms should also be collected, modeled, coded into the simulator, and used to verify ”realistic” results from the simulator. Once proper realism has been verified, mitigation techniques should also be looked into. Again, this should begin with gathering data from current mitigation techniques, coding those techniques into the simulator, and verifying ”realistic” effects of the mitigation mechanism. From there, new mitigation techniques or variations / combinations of current ones may be simulated and analyzed. Further enhancements to the simulator, including these future directions, may be found in Appendix B.4.

## A WPI Topology for Simulation

Described here is the topology used for all simulations in this thesis. It is based in part on [wpi04] and communications with the Network Operations group at WPI.

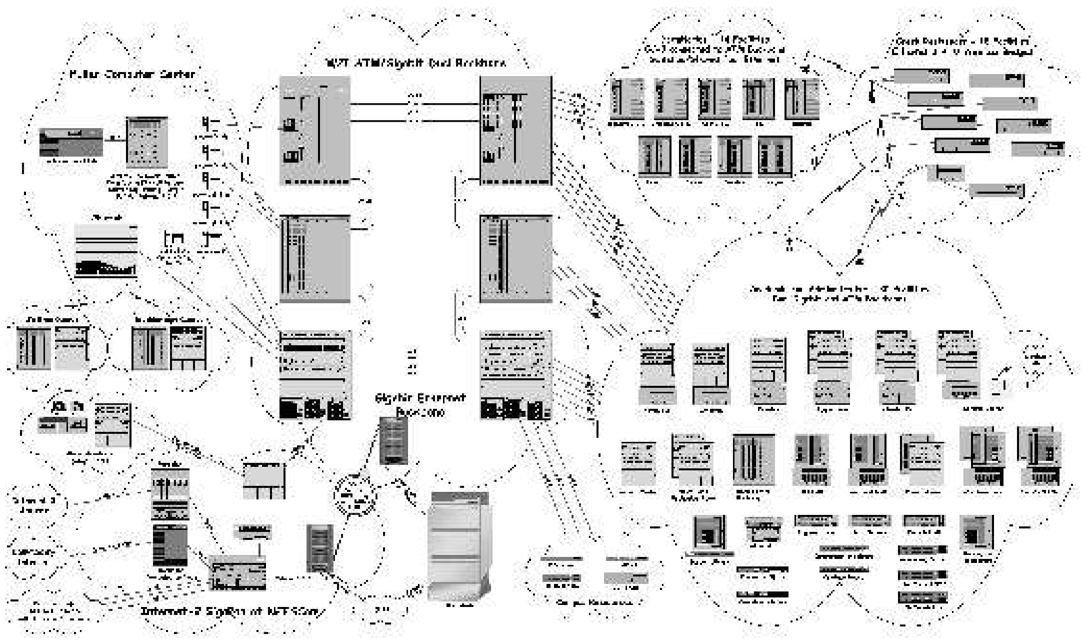
Fuller Computer Center
WPI ATM/Gigabit Dual Backbone
Gigabit Ethernet Backbone
Internet-2 Gigabit at NETSCom/

Figure 11: Campus Network Topology [wpi04]

Provided below is the basic DML configuration file used for all simulations. Such things as up-link bandwidth and worm parameters were varied. However, the structure remained the same.

The main part of the WPI LAN is a modeled as a "backbone" consisting of six routers connected in a ring topology, four 1gig lines between each. One of these routers is then connected to the modeled "Internet" network. The other five have a single network connected to each. Two of these networks consist of 200 hosts, one of 800 hosts, one of 245 hosts, and one of 270 hosts. This is equivalent to 1715 hosts in the total modeled WPI LAN. For most of the simulations, the subnet of 270 hosts also contains the total susceptible population. For the simulations with split susceptible population, the other half of the susceptible hosts were in the subnet of 245 hosts.

## A.1 wpi_reference.dml

```
#  wpi.dml
#
#  Author:  frank p - fspoz3@cs.wpi.edu
#
#  Configuration file used for my thesis.  It models a worm infection using
#  the mixed-abstraction mode as proposed by Liljenstam et. al. and
#  initially encoded into the SSF.App.Worm code.
#  A single LAN is simulated on the packet level, while the worm epidemic
#  in simulated more abstractly (iterating over the epidemic equations) on
#  the Internet level.

# for validation
_schema [_find .schemas.Net]

Net [

  frequency 1000000000    # 1 nanosecond time resolution

  # these are needed, from what I can tell.
  AS_status boundary
  ospf_area 0

  # worm parameters
  worm_model [
    Epidemic [
      _extends .dictionary.worm_model.Slammer

      # Required Attribues:
      #s_0       20      # inital number of suscepible hosts
                         #   in the whole of the Internet.
      #i_0        1      # initial number of infected hosts.
      #beta  1.2e-9      # infection parameter.
    ]
    # Optional Attributes:
```

```
  delta_t     1               # set the time step size (in seconds).
  stratified_on   false       # stratified model (true or false).
  #as_graph    ex_as_topology.adj  # name of AS graph file (connectivity).
                              #   Required for stratified model.
  debug   true                # debug output for global epidemic process?

  # For the MobileCode add-on, part of my thesis (optional):
  #useTarget      SSF.App.Worm.WormLocalBias  # enable for a local bias
  #TargetAcquisition [        # May be used by the TargetAcquisition class
     # What percentage is local bias?
  #  localBias     0.5        # used by WormLocalBias, default 0.5
     # What is the sence of locallity?  This is how many bits to
     # mask the infected host's ip address by, and thus how many bits
     # to use in picking a random number to add to the masked address.
  #  netmask       8          # used by WormLocalBias, default 8
  #]
  #lan_affect_global  true        # enable for LAN feedback to Internet
]

# the PRNG seed value is specified by the 'stream' parameter here
randomstream [
  generator "MersenneTwister"
  stream "seedstartingstring1234567890"
  reproducibility_level "timeline"
]

# the LAN being modelled on the packet level
Net [ id 0

  Net [ id 0
    _extends .networks.backbone.Net
  ]

  Net [ id 1
    _extends .dictionary.net2dorm100
  ]
  link [attach 0:1(9) attach 1:0(0) delay 0.0005]

  Net [ id 2
    _extends .dictionary.net2dorm100
  ]
  link [attach 0:0(9) attach 2:0(0) delay 0.0005]
  Net [ id 3
    _extends .networks.net8dorm100
  ]
  link [ attach 0:5(9) attach 3:0(0) delay 0.0005 ]

  Net [ id 4
    _extends .networks.ak240
  ]
  link [ attach 0:2(9) attach 4:0(0) delay 0.0001 ]

  Net [ id 5
    _extends .networks.fuller
  ]
  link [ attach 0:3(9) attach 5:0(0) delay 0.0001 ]

  # uplink router for connecting to the Internet
  router [ id 0
    _find .dictionary.wormRouterGraph.graph
    # uplink to Internet
    interface [id 0 buffer 8000 _extends .dictionary.15Mb
      monitor [
        # this monitor is required for LAN feedback to Internet
        use SSF.App.Worm.WormMonitorInterfaceUtil
        probe_interval 1.0
        debug true
      ]
    ]
    # link to LAN
    interface [id 1 buffer 8000 _extends .dictionary.15Mb
      monitor [
        # this monitor is good extra info to have
        use SSF.App.Worm.MonitorInterfaceUtil
        probe_interval 1.0
        debug true
      ]
    ]
  ]
  link [ attach 0:4(9) attach 0(1) delay 0.0001 ]
] # end of packet-level LAN

# the Internet network
Net [ id 100
  _extends .networks.theInternet
  ]
  # connect LAN to Internet with a 30ms delay on the wire
  link [ attach 100:0(0) attach 0:0(0) delay 0.03 ]

] # end of Net configuration

# -=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
# a dictionary of entities for referencing / including
dictionary [

  # worm models
  worm_model [
    Epidemic1 [
      s_0     359999
      i_0          1
      beta  1.235e-9
    ]
    Epidemic2 [
      use SSF.App.Worm.StochasticWormEpidemic
      _extends .dictionary.worm_model.Epidemic1
    ]
    Slammer [
      s_0   75000
      i_0   10
      beta  9.313e-7    # for n=4000/s, b=n/(2^32)
    ]
  ]

  # standard network interfaces
  10BaseT [
    bitrate 10000000
    latency 0.0001
  ]
  100BaseT [
    bitrate 100000000
    latency 0.0001
  ]
  45Mb [
    bitrate 45000000
    latency 0.0001
  ]
  50Mb [
    bitrate 50000000
  latency 0.0001
]
15Mb [
  bitrate 15000000
  latency 0.0001
]
10Mb [
  bitrate 10000000
  latency 0.0001
]
1Mb [
  bitrate 1000000
  latency 0.0001
]
300Kb [
  bitrate 300000
  latency 0.0001
]
100Mb [
  bitrate 100000000
  latency 0.0001
]
20Mb [
  bitrate 20000000
  latency 0.0005
]
1Gb [
  bitrate 1000000000
  latency 0.000001
]

# some tcp initialization parameters, as taken from an SSFNet example
tcpinit[
  ISS 10000               # initial sequence number
  MSS 1000                # maximum segment size
  RcvWndSize  32          # receive buffer size
  SendWndSize 32          # maximum send window size
  SendBufferSize 128      # send buffer size
  MaxRexmitTimes 12       # maximum retransmission times before drop
  TCP_SLOW_INTERVAL 0.5   # granularity of TCP slow timer
  TCP_FAST_INTERVAL 0.2   # granularity of TCP fast(delay-ack) timer
  MSL 60.0                # maximum segment lifetime
  MaxIdleTime 600.0       # maximum idle time for drop a connection
  delayed_ack false       # delayed ack option
  fast_recovery true      # implement fast recovery algorithm
  show_report true        # print a summary connection report
] # end of tcpinit

# a router
router3 [
  _find .dictionary.routerGraph.graph
  interface [ id 0 buffer 8000 _extends .dictionary.100BaseT]
  interface [ idrange [from 1 to 2] buffersize 16000]
  #nhi_route [dest default interface 0]
]

# a network with two hosts connected individually to a router
net2 [
  router [ id 0
    _extends .dictionary.router3
  ]
  host [ id 1
    _extends .dictionary.nonsusceptibleHost
    route [dest default interface 0]
  ]
  host [ id 2
    _extends .dictionary.nonsusceptibleHost
    route [dest default interface 0]
  ]
  link [attach 0(1) attach 1(0) delay 0.001]
  link [attach 0(2) attach 2(0) delay 0.001]
] # end of net2

# a network with one router and 100 hosts.  The hosts are in two groups
#   of 50, with the groups connected to one segment with an interface on
#   the router.
net100 [
  router [ id 0
    _extends .dictionary.router3
  ]

  host [
    idrange [from 1 to 50]
    # 2004-07-28:  frankp - we need 'nhi_route' instead of just 'route'
    #   here because we are specifying a next hop... because there's more
    #   than one interface attached to the same link
    _extends .dictionary.nonsusceptibleHost
    nhi_route [dest default interface 0 next_hop 0(1)]
  ]
  host [
    idrange [from 51 to 100]
    nhi_route [dest default interface 0 next_hop 0(2)]
    _extends .dictionary.nonsusceptibleHost
  ]

  link [ delay 0.0005
    attach 0(1)
    attach 1(0) attach 2(0) attach 3(0) attach 4(0) attach 5(0)
    attach 6(0) attach 7(0) attach 8(0) attach 9(0) attach 10(0)
    attach 11(0) attach 12(0) attach 13(0) attach 14(0) attach 15(0)
    attach 16(0) attach 17(0) attach 18(0) attach 19(0) attach 20(0)
    attach 21(0) attach 22(0) attach 23(0) attach 24(0) attach 25(0)
    attach 26(0) attach 27(0) attach 28(0) attach 29(0) attach 30(0)
    attach 31(0) attach 32(0) attach 33(0) attach 34(0) attach 35(0)
    attach 36(0) attach 37(0) attach 38(0) attach 39(0) attach 40(0)
    attach 41(0) attach 42(0) attach 43(0) attach 44(0) attach 45(0)
    attach 46(0) attach 47(0) attach 48(0) attach 49(0) attach 50(0)
  ]
  link [ delay 0.0005
    attach 0(2)
    attach 51(0) attach 52(0) attach 53(0) attach 54(0) attach 55(0)
    attach 56(0) attach 57(0) attach 58(0) attach 59(0) attach 60(0)
    attach 61(0) attach 62(0) attach 63(0) attach 64(0) attach 65(0)
    attach 66(0) attach 67(0) attach 68(0) attach 69(0) attach 70(0)
    attach 71(0) attach 72(0) attach 73(0) attach 74(0) attach 75(0)
    attach 76(0) attach 77(0) attach 78(0) attach 79(0) attach 80(0)
    attach 81(0) attach 82(0) attach 83(0) attach 84(0) attach 85(0)
    attach 86(0) attach 87(0) attach 88(0) attach 89(0) attach 90(0)
    attach 91(0) attach 92(0) attach 93(0) attach 94(0) attach 95(0)
    attach 96(0) attach 97(0) attach 98(0) attach 99(0) attach 100(0)
  ]
] # end of net100

# a network with one router and 20 hosts.  The hosts are in two groups
#   of 10, with the groups connected to one segment with an interface on
#   the router.
net20 [
  router [ id 0
    _extends .dictionary.router3
  ]

  host [
    idrange [from 1 to 10]
    # 2004-07-28:  frankp - for some reason, we need
    #   'nhi_route' instead of just 'route' here...
    nhi_route [dest default interface 0 next_hop 0(1)]
    _extends .dictionary.nonsusceptibleHost
  ]
  host [
    idrange [from 11 to 20]
    nhi_route [dest default interface 0 next_hop 0(2)]
    _extends .dictionary.nonsusceptibleHost
  ]

  link [ delay 0.0005
    attach 0(1)
    attach 1(0) attach 2(0) attach 3(0) attach 4(0) attach 5(0)
    attach 6(0) attach 7(0) attach 8(0) attach 9(0) attach 10(0)
  ]
  link [ delay 0.0005
    attach 0(2)
    attach 11(0) attach 12(0) attach 13(0) attach 14(0) attach 15(0)
    attach 16(0) attach 17(0) attach 18(0) attach 19(0) attach 20(0)
  ]
] # end net20

# a router connected to two routers with connections to 100 hosts each
net2x100 [
  router [ id 0
    _extends .dictionary.router3
  ]

  Net [ id 0
    _extends .dictionary.net100
  ]
  Net [ id 1
    _extends .dictionary.net100
  ]

  link [attach 0(1) attach 0:0(0) delay 0.001]
  link [attach 0(2) attach 1:0(0) delay 0.001]
] # end net2x100
```

```
#
net2dorm100 [
  router [ id 0
    _extends .dictionary.router3
  ]

  Net [ id 0
    _extends .networks.dorm100
  ]
  Net [ id 1
    _extends .networks.dorm100
  ]

  link [ attach 0(1) attach 0:0(0) delay 0.001 ]
  link [ attach 0(2) attach 1:0(0) delay 0.001 ]
] # end net2dorm100

# used for monitoring an interface
probeSession [
  ProtocolSession [name probe use SSF.OS.ProbeSession
    file "rtr_queuedata"
    stream rtrstream
  ]
]

# the protocol graph for a standard router
routerGraph [
  graph [
    ProtocolSession [name ip use SSF.OS.IP]
    ProtocolSession [name ospf use SSF.OS.OSPF.sOSPF]
  ]
] # end routerGraph
# the protocol graph for a LAN's gateway router, that which connects
#   to the Internet and all Internet-LAN traffic must cross through
wormRouterGraph [
  graph [
    ProtocolSession [
      name GatewayProtocolSession
      use SSF.App.Worm.GatewayProtocolSession
      # this works since there is one gateway router
      gateway_id_from_id  true
    ]
```

```
      _extends .dictionary.routerGraph.graph
      _extends .dictionary.probeSession
#      debug true
    ]
  ] # end wormRouterGraph

  # socket communication protocol graph
  sockCommGraph [
    ProtocolSession [ name socket use SSF.OS.Socket.socketMaster ]
    ProtocolSession [ name tcp    use SSF.OS.TCP.tcpSessionMaster
                      warn false
    ]
    ProtocolSession [ name udp    use SSF.OS.UDP.udpSessionMaster
      _find .dictionary.udpinit
                    # warm false
    ]
    ProtocolSession [name ip use SSF.OS.IP]
  ] # end sockCommGraph
  # initialization parameters for the udp protocol
  udpinit [
    max_datagram_size   1024
    debug               false
  ] # end updinit

  # protocol graph for Internet-level worm responder
  inetResponderGraph [graph [
    ProtocolSession [
      name malware_responder
      use SSF.App.Worm.InternetWormResponder

      debug     true
      _find .dictionary.wormConfig.Slammer.protocol
      _find .dictionary.wormConfig.Slammer.port
    ]
    _extends .dictionary.sockCommGraph
  ] ] # end inetResponderGraph

  # protocol graph for Internet-level worm infecter
  inetInfecterGraph [graph [
    ProtocolSession [
      name mobilecode_infecter
      use SSF.App.Worm.InternetWormInfecter
    debug     false
    _find .dictionary.wormConfig.Slammer.protocol
    _find .dictionary.wormConfig.Slammer.port
  ]
  _extends .dictionary.sockCommGraph
] ] # end inetInfecterGraph

# configurations for worms
wormConfig [
  Slammer [
    wormProto [
      name W32fp
      use SSF.App.Worm.SoftwareProtocolSession
      debug       false
      version     1.0
      config      a
      classname   SSF.App.Worm.WormProcess
    ]
    protocol  udp
    port      1434
    size      376
  ]
] # end wormConfig

# the default interface and protocol graph for a non-susceptible host
nonsusceptibleHost [
  interface [id 0 _extends .dictionary.10BaseT]
  graph [ _extends .dictionary.sockCommGraph ]
]

# the default interface and protocol graph for a susceptible host
susceptibleHost [
 interface [id 0 _extends .dictionary.10BaseT]
 graph [
  cpudelay    true
  ProtocolSession [
    name infectable_service
    use SSF.App.Worm.SoftwareProtocolSession
    debug     false
    version   1.0
    config    a
    classname SSF.App.Worm.NetServiceProcess
    net_service [
        _find .dictionary.wormConfig.Slammer.port
        _find .dictionary.wormConfig.Slammer.protocol
      ]
    ]
    ProtocolSession [ _extends .dictionary.operatingSystems.winNT4SP6x86 ]
    ProtocolSession [
      name worm
      use SSF.App.Worm.WormProtocolSession
      debug false
    ]
    _extends .dictionary.sockCommGraph
   ]
  ] # end susceptibleHost

  # operating system specifications
  operatingSystems [
    winNT4SP6x86 [
      name os
      use SSF.App.Worm.OSProtocolSession
      debug         true
      type          Windows
      version       NT4SP6
      architecture  x86
    ]
  ]

  # the default interface and protocol graph for an initially infected host
  infectedHost [
   _find .dictionary.susceptibleHost.interface
   graph [
    ProtocolSession [ _extends .dictionary.wormConfig.Slammer.wormProto ]
    _extends .dictionary.susceptibleHost.graph
   ]
  ]

] # end of dictionary

#-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

# similar to dictionary, a set of networks for reference / including
networks [

    # based on the Fuller Laboratories at WPI
# a network with four subnetworks (equiv to four floors)
fuller [
  router [ id 0
    interface [ id 0 _extends .dictionary.1Gb ]
    interface [ idrange [from 1 to 5] _extends .dictionary.1Gb ]
    _find .dictionary.routerGraph.graph
  ]

  # server room
  Net [ id 0
    _extends .networks.fuller_servers.Net
  ]
  link [ attach 0(5) attach 0:0(0) delay 0.0001 ]

  Net [ idrange [from 1 to 3]
    _extends .networks.ak80
  ]
  # second + third floors
  link [ attach 0(1) attach 1:0(0) delay 0.0001 ]
  # first floor + basement
  link [ attach 0(2) attach 2:0(0) delay 0.0001 ]
  # subbasement
  link [ attach 0(3) attach 3:0(0) delay 0.0001 ]
]
# the Fuller Labs server room
fuller_servers [
  Net [
    # router to connect to
    router [ id 0
      interface [ id 0 _extends .dictionary.1Gb ]
      interface [ idrange [from 1 to 5] _extends .dictionary.100Mb ]
      _find .dictionary.routerGraph.graph
    ]
    # main ccc machines
    host [ idrange [from 1 to 10]
      _extends .dictionary.nonsusceptibleHost
      nhi_route [dest default interface 0 next_hop 0(1)]
    ]
    link [ delay 0.0001
      attach 0(1)
      attach 1(0) attach 2(0) attach 3(0) attach 4(0) attach 5(0)
      attach 6(0) attach 7(0) attach 8(0) attach 9(0) attach 10(0)
    ]
```

```
      # initially infected machine
      #host [ id 11
      #  _extends .dictionary.infectedHost
      #  nhi_route [dest default interface 0 next_hop 0(4)]
      #]
      # other services
      host [ idrange [from 11 to 20]
        _extends .dictionary.susceptibleHost
        nhi_route [dest default interface 0 next_hop 0(4)]
      ]
      link [ delay 0.0001
        attach 0(4)
        attach 11(0) attach 12(0) attach 13(0) attach 14(0) attach 15(0)
        attach 16(0) attach 17(0) attach 18(0) attach 19(0) attach 20(0)
      ]
    ]
  ]

#-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

  # atwater kent
  #   240 hosts (though 480 in reality?)
  #   split by three floors of 80 each, with one router on each floor
  #   all machines on floor connected to floor router
  #   floor routers connected to building router
  ak240 [
    router [ id 0
      interface [id 0 _extends .dictionary.1Gb]
      interface [idrange [from 1 to 4] _extends .dictionary.1Gb]
      _find .dictionary.routerGraph.graph
    ]

    Net [ idrange [from 1 to 3] _extends .networks.ak80 ]
    Net [ id 4 _extends .networks.ak_servers ]

    link [ attach 0(1) attach 1:0(0) delay 0.0001 ]
    link [ attach 0(2) attach 2:0(0) delay 0.0001 ]
    link [ attach 0(3) attach 3:0(0) delay 0.0001 ]
    link [ attach 0(4) attach 4:0(0) delay 0.0001 ]
  ] # end ak240
  # the AK server room
  ak_servers [
    router [ id 0
      interface [ id 0 _extends .dictionary.1Gb ]
      interface [ idrange [from 1 to 5] _extends .dictionary.1Gb ]
      _find .dictionary.routerGraph.graph
    ]

    host [ idrange [from 1 to 5]
      _find .dictionary.nonsusceptibleHost.graph
      interface [ id 0 _extends .dictionary.100Mb ]
      route [dest default interface 0]
    ]
    link [ attach 0(1) attach 1(0) delay 0.0001 ]
    link [ attach 0(2) attach 2(0) delay 0.0001 ]
    link [ attach 0(3) attach 3(0) delay 0.0001 ]
    link [ attach 0(4) attach 4(0) delay 0.0001 ]
    link [ attach 0(5) attach 5(0) delay 0.0001 ]
  ]
  # a set of 80 hosts
  ak80 [
    router [ id 0
      interface [ id 0 _extends .dictionary.1Gb ]
      interface [ id 1 _extends .dictionary.1Gb ]
      _find .dictionary.routerGraph.graph
    ]

    host [ idrange [from 1 to 80]
      _extends .dictionary.nonsusceptibleHost
      nhi_route [dest default interface 0 next_hop 0(1)]
    ]

    link [ delay 0.0001
      attach 0(1)
      attach 1(0) attach 2(0) attach 3(0) attach 4(0) attach 5(0)
      attach 6(0) attach 7(0) attach 8(0) attach 9(0) attach 10(0)
      attach 11(0) attach 12(0) attach 13(0) attach 14(0) attach 15(0)
      attach 16(0) attach 17(0) attach 18(0) attach 19(0) attach 20(0)
      attach 21(0) attach 22(0) attach 23(0) attach 24(0) attach 25(0)
      attach 26(0) attach 27(0) attach 28(0) attach 29(0) attach 30(0)
      attach 31(0) attach 32(0) attach 33(0) attach 34(0) attach 35(0)
      attach 36(0) attach 37(0) attach 38(0) attach 39(0) attach 40(0)
      attach 41(0) attach 42(0) attach 43(0) attach 44(0) attach 45(0)
      attach 46(0) attach 47(0) attach 48(0) attach 49(0) attach 50(0)
      attach 51(0) attach 52(0) attach 53(0) attach 54(0) attach 55(0)
      attach 56(0) attach 57(0) attach 58(0) attach 59(0) attach 60(0)
      attach 61(0) attach 62(0) attach 63(0) attach 64(0) attach 65(0)
      attach 66(0) attach 67(0) attach 68(0) attach 69(0) attach 70(0)
      attach 71(0) attach 72(0) attach 73(0) attach 74(0) attach 75(0)
      attach 76(0) attach 77(0) attach 78(0) attach 79(0) attach 80(0)
    ]
  ] # end ak80

#-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

  # 14 frat networks, of 20 hosts each.
  # two routers (id 0, id 1) with 20Mb interfaces (id 0) open for
  #   external connection
  net14xfrat20 [
    router [ idrange [from 0 to 1]
      interface [id 0 _extends .dictionary.20Mb ]
      interface [idrange [from 1 to 14] _extends .dictionary.20Mb ]
      _find .dictionary.routerGraph.graph
    ]

    Net [ idrange [from 0 to 13]
      _extends .networks.frat20
    ]

    link [ attach 0(1) attach 0:0(0) delay 0.0005 ]
    link [ attach 0(1) attach 1:0(0) delay 0.0005 ]
    link [ attach 0(1) attach 2:0(0) delay 0.0005 ]
    link [ attach 0(1) attach 3:0(0) delay 0.0005 ]
    link [ attach 0(1) attach 4:0(0) delay 0.0005 ]
    link [ attach 0(1) attach 5:0(0) delay 0.0005 ]
    link [ attach 0(1) attach 6:0(0) delay 0.0005 ]
    link [ attach 1(1) attach 7:0(0) delay 0.0005 ]
    link [ attach 1(1) attach 8:0(0) delay 0.0005 ]
    link [ attach 1(1) attach 9:0(0) delay 0.0005 ]
    link [ attach 1(1) attach 10:0(0) delay 0.0005 ]
    link [ attach 1(1) attach 11:0(0) delay 0.0005 ]
    link [ attach 1(1) attach 12:0(0) delay 0.0005 ]
    link [ attach 1(1) attach 13:0(0) delay 0.0005 ]

  ] # end net14xfrat20

#-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
    frat20 [
      Net [
        router [ id 0
          interface [ id 0 _extends .dictionary.20Mb ]
          interface [ idrange [ from 1 to 25 ] _extends .dictionary.10Mb ]
          _find .dictionary.routerGraph.graph
        ]

        host [
          idrange [from 1 to 25]
          _find .dictionary.client100Mb.interface
          _find .dictionary.client100Mb.graph
          nhi_route [dest default interface 0 next_hop 0(1)]
        ]

        link [ delay 0.001
          attach 0(1)
          attach 1(0) attach 2(0) attach 3(0) attach 4(0) attach 5(0)
          attach 6(0) attach 7(0) attach 8(0) attach 9(0) attach 10(0)
          attach 11(0) attach 12(0) attach 13(0) attach 14(0) attach 15(0)
          attach 16(0) attach 17(0) attach 18(0) attach 19(0) attach 20(0)
          attach 21(0) attach 22(0) attach 23(0) attach 24(0) attach 25(0)
        ]
      ]
    ] # end frat20

  #-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

# This is the "dorms" network.  There are 8 dorms of 100 hosts each,
#   which does not match the WPI network, but approximates it.
# One router (id 0) has an open interface (id 0) for external connection.
#   It also has an interface for connection to the frat network, to simulate
#   the wireless connection that they have.
    net8dorm100 [
      # main external connection router, with interface 0 open,
      #   interface 1 to 8 for internal networks,
      #   and interface 11 open for frat connection
      router [ id 0
        interface [ id 0 _extends .dictionary.1Gb ]
        interface [ idrange [from 1 to 8] _extends .dictionary.1Gb ]
        interface [ id 11 _extends .dictionary.100Mb ]
        _find .dictionary.routerGraph.graph
      ]
    Net [ idrange [from 0 to 7]
      _extends .networks.dorm100
    ]
    link [ delay 0.0001 attach 0(1) attach 0:0(0) ]
    link [ delay 0.0001 attach 0(2) attach 1:0(0) ]
    link [ delay 0.0001 attach 0(3) attach 2:0(0) ]
    link [ delay 0.0001 attach 0(4) attach 3:0(0) ]
    link [ delay 0.0001 attach 0(5) attach 4:0(0) ]
    link [ delay 0.0001 attach 0(6) attach 5:0(0) ]
    link [ delay 0.0001 attach 0(7) attach 6:0(0) ]
    link [ delay 0.0001 attach 0(8) attach 7:0(0) ]
  ]

#-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

  # routers 0 thru 5, attached in a ring formation.
  # each router is connected to two other routers by four
  #   interfaces each.
  # this leaves interfaces 9 thru 15 on each router open for
  #   attachment to other nets.
  backbone [
    Net [

      router [ idrange [from 0 to 5]
        interface [ idrange [from 0 to 11] _extends .dictionary.1Gb ]
        #_find .dictionary.routerGraph.graph
        _find .dictionary.routerGraph.graph
        route [dest default interface 0]
      ]

      # router 0 attach to router 1
      link [ attach 0(0) attach 1(4) delay 0.0001 ]
      link [ attach 0(1) attach 1(5) delay 0.0001 ]
      link [ attach 0(2) attach 1(6) delay 0.0001 ]
      link [ attach 0(3) attach 1(7) delay 0.0001 ]
      # router 1 attach to router 2
      link [ attach 1(0) attach 2(4) delay 0.0001 ]
      link [ attach 1(1) attach 2(5) delay 0.0001 ]
      link [ attach 1(2) attach 2(6) delay 0.0001 ]
      link [ attach 1(3) attach 2(7) delay 0.0001 ]
      # router 2 attach to router 3
      link [ attach 2(0) attach 3(4) delay 0.0001 ]
      link [ attach 2(1) attach 3(5) delay 0.0001 ]
      link [ attach 2(2) attach 3(6) delay 0.0001 ]
      link [ attach 2(3) attach 3(7) delay 0.0001 ]
      # router 3 attach to router 4
      link [ attach 3(0) attach 4(4) delay 0.0001 ]
      link [ attach 3(1) attach 4(5) delay 0.0001 ]
      link [ attach 3(2) attach 4(6) delay 0.0001 ]
      link [ attach 3(3) attach 4(7) delay 0.0001 ]
      # router 4 attach to router 5
      link [ attach 4(0) attach 5(4) delay 0.0001 ]
      link [ attach 4(1) attach 5(5) delay 0.0001 ]
      link [ attach 4(2) attach 5(6) delay 0.0001 ]
      link [ attach 4(3) attach 5(7) delay 0.0001 ]
      # router 5 attach to router 0
      link [ attach 5(0) attach 0(4) delay 0.0001 ]
      link [ attach 5(1) attach 0(5) delay 0.0001 ]
      link [ attach 5(2) attach 0(6) delay 0.0001 ]
      link [ attach 5(3) attach 0(7) delay 0.0001 ]

    ]
  ] # end of backbone

#-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

# A dorm with four lans of 25 hosts each with 10Mb connections
  #   and a 1Gb external connector
  dorm100 [
      router [ id 0
        interface [ id 0 _extends .dictionary.1Gb ]
        interface [ id 1 _extends .dictionary.1Gb ]
        interface [ idrange [from 2 to 5] _extends .dictionary.100Mb ]
        _find .dictionary.routerGraph.graph
      ]

      host [
        idrange [from 1 to 25]
        _find .dictionary.nonsusceptibleHost.interface
        _find .dictionary.nonsusceptibleHost.graph
        nhi_route [dest default interface 0 next_hop 0(2)]
      ]
      host [
        idrange [from 26 to 50]
        _find .dictionary.nonsusceptibleHost.interface
  _find .dictionary.nonsusceptibleHost.graph
  nhi_route [dest default interface 0 next_hop 0(3)]
]
host [
  idrange [from 51 to 75]
  _find .dictionary.nonsusceptibleHost.interface
  _find .dictionary.nonsusceptibleHost.graph
  nhi_route [dest default interface 0 next_hop 0(4)]
]
host [
  idrange [from 76 to 100]
  _find .dictionary.nonsusceptibleHost.interface
  _find .dictionary.nonsusceptibleHost.graph
  nhi_route [dest default interface 0 next_hop 0(5)]
]

link [ delay 0.0007
  attach 0(2)
  attach 1(0) attach 2(0) attach 3(0) attach 4(0) attach 5(0)
  attach 6(0) attach 7(0) attach 8(0) attach 9(0) attach 10(0)
  attach 11(0) attach 12(0) attach 13(0) attach 14(0) attach 15(0)
  attach 16(0) attach 17(0) attach 18(0) attach 19(0) attach 20(0)
  attach 21(0) attach 22(0) attach 23(0) attach 24(0) attach 25(0)
]
link [ delay 0.0007
  attach 0(3)
  attach 26(0) attach 27(0) attach 28(0) attach 29(0) attach 30(0)
  attach 31(0) attach 32(0) attach 33(0) attach 34(0) attach 35(0)
  attach 36(0) attach 37(0) attach 38(0) attach 39(0) attach 40(0)
  attach 41(0) attach 42(0) attach 43(0) attach 44(0) attach 45(0)
  attach 46(0) attach 47(0) attach 48(0) attach 49(0) attach 50(0)
]
link [ delay 0.0007
  attach 0(4)
  attach 51(0) attach 52(0) attach 53(0) attach 54(0) attach 55(0)
  attach 56(0) attach 57(0) attach 58(0) attach 59(0) attach 60(0)
  attach 61(0) attach 62(0) attach 63(0) attach 64(0) attach 65(0)
  attach 66(0) attach 67(0) attach 68(0) attach 69(0) attach 70(0)
  attach 71(0) attach 72(0) attach 73(0) attach 74(0) attach 75(0)
]
link [ delay 0.0007
  attach 0(5)
  attach 76(0) attach 77(0) attach 78(0) attach 79(0) attach 80(0)
```

```
          attach 81(0) attach 82(0) attach 83(0) attach 84(0) attach 85(0)
          attach 86(0) attach 87(0) attach 88(0) attach 89(0) attach 90(0)
          attach 91(0) attach 92(0) attach 93(0) attach 94(0) attach 95(0)
          attach 96(0) attach 97(0) attach 98(0) attach 99(0) attach 100(0)
        ]
    ]  #end dorm100

  #-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-==

  # The MobileCode stuff needs an "Internet" net to act as the translation
  #   between macroscopic level and packet level.
  theInternet [
    # the connecting router
    router [ id 0
      interface [ idrange [from 0 to 2] _extends .dictionary.100Mb ]

      _find .dictionary.routerGraph.graph
    ]

    # host which sends infections into the LAN
    host [ id 1
      interface [ id 0 _extends .dictionary.100Mb ]
      route [dest default interface 0]
      _find .dictionary.inetResponderGraph.graph
    ]

    # host which receives infections from the LAN
    host [ id 2
      interface [ id 0 _extends .dictionary.100Mb ]
      route [dest default interface 0]
      _find .dictionary.inetInfecterGraph.graph
      graph [ _extends .dictionary.sockCommGraph ]
    ]

    # linking the hosts to the router
    link [ attach 0(1) attach 1(0) delay 0.0001 ]
    link [ attach 0(2) attach 2(0) delay 0.0001 ]

  ]  #end theInternet

  #-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-==

]
```

# B Using the Simulator

The simulator used in this work was based on SSFNet [ssf04], a packet-level network simulator maintained by the Renesys Corporation, with contribution of the original SSF.App.Worm framework by Michael Liljenstam. The latest versions of both may be found on their respective websites, `http://www.ssfnet.org` and `http://www.crhc.uiuc.edu/m̃ili/research/ssf/worm/`. The code extensions mentioned here were developed using *SSFNet v2.0* and *SSF.App.Worm v0.6*.

## B.1 Specific modifications

The following files were created or modified to incorporate the worm framework as described in Section 5.2.

**WormEpidemicState.java** Added a variable for affecting the infection rate. This is used by the feedback mechanism described in this work.

**DeterministicWormEpidemicState.java and StochasticWormEpidemicState.java** Added adjustment to the calculation of *BetaJ* in the *update()* function, $BetaJ* = infection\_adjuster$, to use the WormEpidemicState variable for affecting the infection rate.

**MacroscopicModelConfigurator.java** Added variables and code for obtaining and setting new DML attributes in the Epidemic section. Specifically, for the *useTarget*, *TargetAcquisition*, and *lan_affect_global* parameters. Also, bugfix for obtaining the *randomstream* seed in the DML configuration file.

**MacroscopicModel.java** Added code for organizing the interaction between the Internet level and the packet level. Of particular note, one of the changes to the *reg-*

*isterBorderRouter()* function assumes that the GatewayProtocolSession is installed on the router that connects the LAN to the Internet, and that the particular router is in the top-level *Net* for that LAN.

**SSF/Net/lanLinkLayer.java** Added an IndexOutOfRange check to print out an error instead of causing an Exception. This was done so that the simulations would still run even with this error every once in a while. Further investigation of the cause for this error is necessary.

**InternetWormInfecter.java** Implements the protocol session that creates actual packet worm scans from the Internet level to the LAN level. The code assumes there is one instance of this.

**InternetWormResponder.java** Implements the protocol session that receives the packet worm scans from the LAN level to the Internet level. The code assumes there is one instance of this.

**WormMonitorInterfaceUtil.java** Modeled on SSF.App.Worm.MonitorInterfaceUtil. Used to monitor the gateway router's drops and provide feedback for the Macro-scopicModel, in order to adjust the "Internet" scanning rate. This should be set as a monitor on the interface that connects the LAN's gateway router to the Internet.

**OSProtocolSession.java** A pseudo-protocol that models an operating system, such as the type and version, as well as the architecture. Also handles instantiating different software upon "boot" and upon infection.

**SoftwareProtocolSession.java** Converts the DML configuration into an instance of a SoftwarePackage and registers it with the hosts OSProtocolSession.

**SoftwarePackage.java** Represents an installed software package on a system. It is separate from an actual instance of the software running on the system, as represented by the SoftwareProcess class.

**SoftwareProcess.java** The instance of a *SoftwarePackage* running on a host. This is the base class that all software should extend. For instance, *NetServiceProcess* and *WormProcess* both extend this class.

**NetServiceProcess.java** A specific instance of SoftwareProcess that allows setting of a port and specific protocol to listen for incoming worm connections. When a particular worm is transfered to this service over the network, that worm is passed to the OSProtocolSession and "executed" on the host.

**WormProcess.java** A running instance of a particular worm.

**WormTargetAcquisition.java** The base class for a worm's target acquisition method. This class implements the random scanning method. Other target acquisition types should extend this class, such as the *WormLocalBias* class.

**WormLocalBias.java** Extends *WormTargetAcquisition* to provide for a certain local bias in the worm's scanning strategy. May be configured through the TargetAcquisition parameter block in the Epidemic block of the DML configuration file. May be provided with a certain bias, between zero and one, and a bit mask, which enables varying the idea of locality. The default values are $localbias = 0.5$ and $netmask = 8$, equivalent to a 50% local bias within the local class C subnet.

**WormIV.java** The method used to attempt transfer of the worm code to the chosen host. The default is to use Slammer's strategy of sending a single 376 byte

UDP packet.

## B.2 Experience with SSFNet

Problems were encountered while working with the SSFNet simulator. In particular, the error messages were often non-descript Java exceptions. The examples and user documentation on the SSFNet website [ssf04] were sometimes difficult to follow and seemingly incomplete. A single mailing list[9] was found for community support, however did not seem very developed. Overall, the SSFNet simulator and the community around it seems to be in need of further maturing before widespread use will become plausible.

## B.3 Current state of the simulator's capabilities

Many simulation capabilities were proposed in this work. However, due to time constraints and the need to show a proof-of-concept working structure, some details were left out of this first implementation.

Currently, the Slammer worm is implemented within the simulator. This means that the UDP communication, random IP selection, and simple infection vector are implemented and have been tested. There has also been implementation and testing to allow for a local bias IP selection. Also, implementation for a TCP communicating worm, however without testing. It is important to note that some of the Slammer characteristics, such as worm size within the NetServiceProcess, have been hard-coded into the system and should be abstracted out to DML configuration options.

[9]`https://list.eecs.harvard.edu/mailman/listinfo/ssfnet`

## B.4 Proposed road-map for further implementation

We believe the creation of a simulation tool for worm propagation is an important next-step for the worm research community. The framework developed here is a first step toward such an effort. As such, there are many more steps to take in making the currently developed code-base suitable for general use. For example,

1. Move code to SSF.App.MobileCode for separation and easier packaging. The files that were created for this project should be moved out of the SSF.App.Worm tree and into a new tree. I believe that MobileCode is an appropriate name, since it is a framework for code that can propagate over a network. The files which were changed should also be sorted through to see what may be taken out of the SSF.App.Worm tree and put into its own files in the SSF.App.MobileCode tree.

2. The code needs to be cleaned up, especially in terms of removing hard-coded values such as worm size and creating DML configurable parameters.

3. Perform proper verification of the simulator. This should involve gathering information on Slammer propagation within a particular network and the topology of that network. It should also involve more accurate coding of Slammer's "faulty" target acquisition method.

4. Implement and test a simple TCP worm. This should involve the proper verification methods used in the prior task.

5. Implement a scanning strategy.

6. Implement and test simple mitigation techniques. This should involve proper verification of each mitigation technique functionality by setting up a real-

world scenario, gathering data, doing the same in the simulator, and comparing.

7. Brainstorm and implement Goal-Based Actions (GBAs).

8. Brainstorm and implement mitigation techniques.

Version 2.0 should encompass the following functionality:

1. Allow for more general parameters, such as number of susceptible hosts and their distribution in the network. This will involve dynamically "installing" software on hosts at configuration time, which should be tedious but not difficult.

2. Allow for background traffic.

3. Allow for traffic on the same port as the susceptible NetService(s).

4. Allow multiple worms / epidemics at once

## C Selected communications

Provided here are selected e-mail communications obtained while conducting this thesis. We believe the information provided may be useful for future work or reference.

```
Date: Tue, 31 Aug 2004 10:04:40 -0500
From: Michael Liljenstam
To: frank p <fspoz3@WPI.EDU>
Subject: Re: worm propagation data
Parts/Attachments:
   1.1   OK      82 lines  Text
   1.2 Shown     73 lines  Text
   2           286 KB     Application
```

---------------------------------------

Hi Frank -

Sorry about that. It slipped into the background and disappeared. Thanks for reminding me.

The attached tarball replaces the src/SSF/App/Worm directory code. So make sure to move and save any code you've modified before putting this in place. It should be expanded in the ssfnet root directory and also replaces a couple of files of the core ssfnet classes (these classes have been slightly modified).

Look at the GatewayRouterSession code where I look at the inflow of scans to generate ICMPs.

Some comments below on what we discussed before.

Hope that helps,

/Michael

Date: Wed, 15 Sep 2004 09:40:42 -0500 (CDT)
From: Michael Liljenstam
To: frank p <fspoz3@WPI.EDU>
Subject: Re: worm propagation data

Hi Frank -

On Tue, 14 Sep 2004, frank p wrote:

> First, thanks for the beta code.  Do you have anyone actively working on
> this right now?  I'm thinking that I may be able to contribute to the
> codebase by the end of my thesis in December.

It's just me working on this code base, and I tend to get busy working on other things. So I certainly wouldn't mind if you want to contribute :)
We have a grad student also working on worm modeling, but he's using another code base built on top of the C++ version of SSFNet (iSSFNet) using fluid traffic modeling, so it's a bit different; and I'd like to keep this code going 'cause it has some features the other code lacks.

> Second, a query on the calculation of "outbound scan rate" from gateway

```
> routers.  I'm seeing an outbound rate greater than zero even before any
> hosts within the network (AS) behind the router are infected.  It may be
> that I do not have a complete understanding of how the network relates to
> an AS and how the infected machines are calculated.  In any case, I'm
> really perplexed by this, so any light you can shed would be much
> appreciated.

I'll have to check this and get back to you. Possibly it's the case that
the "outbound scand rate" also includes "transit" scans, just passing
through. But I really can't remember. Otherwise, it doesn't sound right.

/Michael

--
Michael Liljenstam, Post-Doctoral Research Associate
Center for Reliable and High-Performance Computing, Coordinated Science Lab.
University of Illinois at Urbana-Champaign, 1308 W. Main St., Urbana, IL 61801
http://www.crhc.uiuc.edu/~mili

Date: Fri, 17 Sep 2004 19:49:36 -0500 (CDT)
From: Michael Liljenstam
To: frank p <fspoz3@WPI.EDU>
Subject: Re: worm propagation data

Hi Frank -

On Thu, 16 Sep 2004, frank p wrote:

> > keep this code going 'cause it has some features the other code lacks.
>
> Just out of curiosity (and 'cause it might look nice to list in my thesis
> report), which features make the SSFNet code unique/useful to you?

Two things come to mind right away:
- SSFNet has quite good routing models, OSPF and BGP. The worm model has
hooks into the routing layers so it will find out when routes change and
can interact with the routing dynamics (it can influence the routing).
We've used this to attempt to model how worms might affect BGP dynamics.
- It has some code to dump packets in tcpdump format that we've used to
generate attack traffic traces for worm detection system testing.

> > > Second, a query on the calculation of "outbound scan rate" from gateway
> > > routers.  I'm seeing an outbound rate greater than zero even before any
> > > hosts within the network (AS) behind the router are infected.  It may be
> > > that I do not have a complete understanding of how the network relates to
> > > an AS and how the infected machines are calculated.  In any case, I'm
> > > really perplexed by this, so any light you can shed would be much
> > > appreciated.
> Hopefully to help clarify (and reduce the time needed looking into it),
> I'm specifically referring to the "two_as_traffic" test in the tests/
> directory.  It seems to show an outgoing scan rate from each router the
> host behind it becomes infected.

Oh, ok, this is quite simple and just has to do with the assumptions of
the model. So, it uses a stratified deterministic epidemic model that spreads
a large the vulnerable population over _all_ ASes. So, think of this as
those two hosts that you see in the two ASes there are just part of a
larger population of vulnerable hosts (supposing that you're particularly
interested in if/when those hosts get infected for some reason). Hence,
there are other hosts that are getting infected in those ASes that are not
explicitly modeled (they're just modeled through a number that describes
the size of the population).

Hope that clarifies things a bit.

Regards,

Michael

Date: Fri, 24 Sep 2004 11:03:22 -0500
From: Michael Liljenstam
To: frank p <fspoz3@WPI.EDU>
Subject: Re: worm propagation data
Parts/Attachments:
   1   OK     39 lines  Text
   2 Shown    42 lines  Text
----------------------------------------

Hi Frank -

frank p wrote:
> So the starting Epidemic Model parameters (#of susceptible,
> #of infected)
> are split evenly between all of the ASes?  Or does each AS
> have a copy of
> the parameters as set in the DML?  Is there any way to
```

> explicitly set
> parameters for one AS and different params for another?

You can accomplish this, but not through DML alone. When writing the code
I recognized that the initial distribution of susceptible/infected hosts
might be a very compilicated function of several factors, so I thought it
would be difficult to have a flexible enough pure DML description. Hence,
the initializer abstract class and different subclasses that implement
different initial distributions. So, you would need to write some code to
implement your own initializer class, modeled after, say, the uniform
initializer.
Try to take a look at the code. It might not be so obvious what it should
do, but you're welcome to come back with more questions and I'll try to
help you along.

Regards,

Michael

Date: Mon, 25 Oct 2004 10:54:11 -0500 (CDT)
From: Michael Liljenstam
To: frank p <fspoz3@cs.WPI.EDU>
Subject: Re: worm simulation questions

Hi Frank -

On Mon, 25 Oct 2004, frank p wrote:
> I have a few questions about the simulator, but first wanted to ask if
> you'll be attending the ACM CCS conference in DC this week.  My advisor
> here at WPI is funding one student from my research group to attend, and
> it looks like I'm the lucky one :+)
> If you'll be there and have some time, I'd love to sit down and talk with
> you about worm simulation for a few minutes.

Unfortunately, I can't make it.

> Now, my questions are more of a general than specific nature.  Basically,
> I'd like to know if you think this makes sense, and how much work it might
> take to make SSFnet simulate something like this:
> - There is an Internet AS, with a bunch of susceptibleand infected hosts.
> - There is one or more subnet ASes that are explicitly defined by the DML
> network structure.  This means that the only susceptibles are the hosts
> which include the WormProtocolSession, and the choice of initially

```
> infected can either be excplicit (flag in each WormProtocolSession) or
> random (given a set number of initially infected, of course).

Sounds straightforward.

> - The interactions between each AS and the Internet are only through the
> GatewayProtocolSession routers, allowing the instantiation of pkts when
> scans come in and inverse when scans go out.

Yes. The only tricky part here is to come up with a reasonable packet
arrival process, i.e. to decide packet arrival instants from the fluid
rate. Shouldn't be much coding work though. There's something similar
to this in the 0.6 code.

> What I'm trying to do is create a worm algorithm that can be used to try
> different combinations of worm parameters (target discovery method,
> infection method, propagation rate limitting, etc.) and show how they
> affect the propagation of a worm within a campus-sized network.  The
> assumption is that there are also incoming infection scans from "The
> Internet", but that the rate of these can be handled by the abstract
> epidemic models.

So, are you planning to simulate scans at packet level within the AS then?
In one AS it should be ok, but there are some minor details that one will
have to deal with. We mentioned some of that in our 2002 paper on mixed
abstraction modeling. Try googling for "packet level simulation of worm"
for a recent paper by Perumalla where he also discusses some of those
things.

Cheers -
Michael

Date: Fri, 07 Jan 2005 13:06:50 -0600
From: Michael Liljenstam
To: frank p <fspoz3@cs.wpi.edu>
Cc: ssfnet@eecs.harvard.edu
Subject: Re: [SSFNet] random numbers reproducibility_level
Parts/Attachments:
   1   OK     39 lines  Text
   2 Shown    35 lines  Text
----------------------------------------

frank p wrote:
```

Can anyone give a good description of what the "reproducibility_level" of SSFNet random numbers means? I’ve read the description on this page:

http://www.ssfnet.org/InternetDocs/ssfnetDMLReference.html

and it seems to me to be saying that there will always be a level of reproducibility.

Yes. "Controlled randomness" is usually what one wants.
The SSFNet "level of reproducibility" lets you control how many random streams you use to generate your random variates. Most textbooks will tell you to use a separate random number stream per distribution you’re sampling from. This will tend to be a _lot_ of streams. This config setting lets you share the same stream over all variates in a protocol, on a host, or in a timeline if you prefer.

Or is there a way to run a simulation that will not produce the same output every time? Maybe I’m not seeing it, but I’d even settle for telling the simulation to seed the PRNG with the current time.

Usually you want to pick a new seed for each replication in some structured fashion. You set it in the DML. If you get some unexpected or strange result from one of your replications, you want to know that you can reproduce it. Thus, picking "current time" or similar is usually avoided. But, as always, it depends on what you want to do...

Hope that helps.

/Michael

# References

[cer04] Cert coordination center. http://www.cert.org/, December 2004.

[hon04] The honeynet project. http://project.honeynet.org/, December 2004.

[Inc03] Network Associates Inc. Description of w32/nachi.worm. http://vil.nai.com/vil/content/v_100559.htm, August 2003.

[Inc04a] Network Associates Inc. Description of w32/bagle.p. http://vil.nai.com/vil/content/v_101098.htm, March 2004.

[Inc04b] Network Associates Inc. Description of w32/mydoom. http://vil.nai.com/vil/content/v_100983.htm, February 2004.

[LNBG03] Michael Liljenstam, David M. Nicol, Vincent H. Berk, and Robert S. Gray. Simulating realistic network worm traffic for worm warning system design and testing. In *Proceedings of the 2003 Workshop on Rapid Malcode (WORM) Washington*, October 2003.

[LYPN02] M Liljenstam, Y Yuan, BJ Premore, and D Nicol. A mixed abstraction model of large-scale internet worm infestations. In *Proceedings of the Tenth IEEE/ACM Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASSCOTS)*, October 2002.

[MPS+03] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. The spread of the saphire/slammer worm. Technical report, A joint effort of CAIDA, ICSI, Silicon Defense, UC Berkeley EECS and UC San Diego CSE, 2003.

[ns2] The Network Simulator - ns-2 Project Homepage: http://www.isi.edu/nsnam/ns/.

[nws] The Network Worm Simulator - NWS Project Homepage: http://www.users.qwest.net/ eballen1/nws/.

[sla04] Internet attack. http://www.cnn.com/2003/TECH/internet/01/25/internet.attack/, March 2004.

[SPW02] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to 0wn the internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium (Security '02)*, 2002.

[ssf] A Network Worm Modelling Package for SSFNet Project Homepage: http://www.crhc.uiuc.edu/ mili/research/ssf/worm/index.html.

[ssf04] Scalable simulation framework. http://www.ssfnet.org/, March 2004.

[Vog03] Tom Vogt. Simulating and optimising worm propagation algorithms. http://web.lemuria.org/security/WormPropagation.pdf, September 2003.

[wor04] Computer worm. http://en.wikipedia.org/wiki/Computer_worm/, October 2004.

[wpi04] Wpi network operations - network infrastructure. http://www.wpi.edu/Admin/Netops/infrastructure.html, March 2004.

[WPSC03] Nicholas Weaver, Vern Paxson, Stuart Staniford, and Robert Cunningham. A taxonomy of computer worms. In *Proceedings of the ACM Workshop on Rapid Malware (WORM)*, 2003.

[ZGT03] Cliff C. Zou, Weigo Gong, and Don Towsley. Worm propagation modeling and analysis under dynamic quarantine defense. In *ACM Workshop on Rapid Malcode*, 2003.

[ZTG03] Cliff C. Zou, Don Towsley, and Weibo Gong. On the performance of internet worm scanning strategies. Technical Report TR-03-CSE-07, University of Massachusetts, Amherst, 2003.

[ZTGC03] Cliff C. Zou, Don Towsley, Weibo Gong, and Songlin Cai. Routing worm: A fast, selective attack worm based on ip address information. Technical Report TR-03-CSE-06, University of Massachusetts, Amherst, 2003.