Evolution of Secure Networks through Simulation

Mason Tang

Junior Class

Mass Academy of Mathematics & Science

March 11, 2005
Abstract

The rapid spread of computer worms and viruses has become a major threat to governments, businesses, educational institutions, and even home computing. In order to combat this threat, a computational procedure based on evolution principles was introduced to create a more secure computer network. A series of computer programs written in Perl was developed and used in conjunction with a Perl package known as the Network Worm Simulator (NWS) to effectively simulate large-scale computer networks and control the evolution. The evolutionary system was run for 100 generations of 10 networks, or 1000 networks, in order to generate the optimized output. The optimized networks showed an average of 50% improvement on average infection rate. In addition, it was found that network infection rate is inversely proportional to the number of generations simulated, and that distributed network topologies lead to more secure networks. Some validation was given by SSFNet under a global Internet environment. However, more model improvements are needed for realistic implementations.
# Table of Contents

Abstract................................................................................................................................2

Table of Contents.................................................................................................................3

1. Introduction......................................................................................................................4
   1.1. Problem Statement............................................................................................5
   1.2. Literature Review..............................................................................................6
   1.3. Significance.......................................................................................................9

2. Models and Methods......................................................................................................10
   2.1. Definitions and Parameters.............................................................................10
   2.2. Model Design Goals .......................................................................................14
   2.3. Program Structure...........................................................................................15

3. Results and Analysis......................................................................................................18
   3.1. Results with Network Worm Simulator..........................................................18
   3.2. Validation Using SSFNet................................................................................22

4. Discussion......................................................................................................................24

5. Conclusions....................................................................................................................25

Acknowledgement .............................................................................................................25

References..........................................................................................................................26

Appendices.........................................................................................................................27
   Appendix A: Programs written for this project
   Appendix B: Source Code
   Appendix C: Description of Simulators
1. Introduction

The introduction of the Internet has revolutionized communication and data storage for virtually every computer user in the world. A secure Internet and computer networks are vital to every branch of the society and every aspect of our daily life where information exchange and communication are conducted through any computer network. Indeed, a malfunction in a computer network belonging to the federal government or a hospital could be disastrous in many cases. Perhaps an even more dangerous scenario would be the compromise of a computer network owned by the military, potentially revealing critical information to enemy eyes. Therefore, the securing and protection of computer networks has become an extremely urgent task in our computerized society.

In this project, we introduce a computational procedure based on natural evolutionary principles to generate more secure computer networks. In order to measure the fitness of the computer networks, we use an objective function defined as the infection rate of each network. Beginning with an initial network, by running the evolution for 100 generations, the output network was generally seen to exhibit a decrease in infection rate by 30% - 50%. As with most problems of this nature, this result can most likely be improved by modifying the algorithms used by the evolution programs.
1.1 Problem Statement

The rapid spread of computer worms and viruses has become a major threat to large computer networks. Creative methods of designing and developing more secure networks to fight virus spread are urgently needed. The goal of this project is to introduce a computational procedure based on natural evolution principles to generate more secured computer networks. By optimizing the structure of networks, we hope to increase the safety and security of computer networks in government, business, schools, industries and our personal lives, and minimize potential damages if a computer worm attack were to occur.

Fig. 1.1 Computer network infection is becoming a threat to government, national security, education, business, industries, and normal everyday life.
1.2 Literature Review

Currently, there have been many attempts at modeling network traffic and worm spread across networks. One particularly relevant example of this is described in a paper written by Dan Ellis, entitled “Worm Anatomy and Model” [1]. This paper summarizes a method for the modeling of worm propagation on networks and makes several general conclusions on the design of worm simulations. The paper describes the general behavior of worms (viruses) on a network and goes on to outline specific methods for modeling the spread of worms. The paper begins by summarizing previous attempts at an analysis of worm spread on a network, and outlines the various shortcomings of previous efforts. Ellis then proceeds to outline the behavior of a worm in terms of phases, such as the target acquisition phase, etc. These phases are critical in the accurate simulation of an artificial computer worm.

Another paper, “Experiences with worm propagation simulations”, written by Arno Wagner, Bernhard Plattner, Thomas Dübendorfer, and Roman Hiestand describes several methods for the simulation of viral spread on large scale networks, such as the Internet [2]. The approaches mentioned included mathematical models, limited test beds, real-world experiments, and most importantly computer simulations. Like the previous paper, this work describes the basic behaviors of viruses in a network environment, and uses these assumptions to develop a system for simulating their behavior. Many of the concepts illustrated here were used in the implementation of the simulation packages, including both SSFNet and NWS. In addition to a description for a model of an Internet-size network, this paper compares different propagation methods for viruses, such as TCP vs. UDP, two methods of packet transfer used on networks, and addresses issues that
affect the spread of viruses across a realistic network. Such factors play an important role in evolving the networks, with some even behaving as parameters controlling the evolution of the network. The paper describes results from a model using both the Code Red and Sapphire viruses, and summarizes the results.

Another attempt at the simulation of worm traffic on networks is described in the paper, “A Mixed Abstraction Level Simulation Model of Large-Scale Internet Worm Infestations,” by Michael Liljenstam, Yougo Yuan, BJ Premore, and David Nicol [3]. This paper focuses on approaches to overcoming the obstacles present in modeling a large-scale infection of a network, such as large size, and long time scales. The authors circumvent this obstacle by limited abstraction, and making a number of assumptions to simplify the computational requirements for the simulation. To simplify the computational requirements for such a model, the simulation occurs at the interdomain level, instead of at an individual host level. The basic units of the simulation, instead of individual hosts, would be ASs, or Autonomous Systems. These ASs model large groups of computers, instead of individual computers, thus reducing computational requirements. This makes simulating large networks, like those in this project, much more efficient without introducing a substantial amount of error.

The majority of the project revolves around the subject of evolution within a computer simulation. Currently, there are two main approaches to the problem of creating an effective evolutionary system in an artificial environment, which will subsequently be referred to as parameter optimization. The first approach uses what are known as Genetic Algorithms (GAs), and were originally developed for combinatorial optimization problems and discrete adaptation. The second approach uses Evolution
Strategies (ESs). The differences and similarities between these two approaches are described in Frank Hoffmeister & Thomas Bäck’s “Genetic Algorithms and Evolution Strategies: Similarities and Differences” [4]. Unlike GAs, ESs are geared toward experimental problems with continuously changeable parameters. Both methods were developed in the 1960s and overlap to a certain degree in their applications. This stems from the fact that both implement the same basic principles of population, mutation, recombination, and selection [4]. For the purposes of this project, an algorithm using an ES is the most viable option. The simulation program more closely resembles an experiment than a discrete function, and thus would be best controlled by an ES.

In addition to work in parameter optimization, there has also been a great deal of research focusing on understanding the Internet as a whole. Currently, this has been done mostly by monitoring traffic through the wide area network. A paper written by Vinod Yegneswaran, Paul Barford, and Johannes Ullrich entitled “Internet intrusions: global characteristics and prevalence” describes a study conducted using firewall logs from hosts all over the world [5]. The results of the analysis concluded that although sources of worms were uniformly spread, there were always a small number of hosts that were responsible for a disproportionately large fraction of the total infection. This project can benefit from data on general trends of worm propagation on the Internet scale, and can use the data to create a better final result.

While many researchers have investigated virus spread and network security, a method that will take an existing realistic network, perform a series of changes based on simulated performance and natural selection, and finally generate a re-structured network with optimized virus-resistant features is lacking in the current literature.
1.3 Significance

In almost any large organization in today’s modern world, one can find some form of computer network. This proliferation of networked computers has led to an increasing vulnerability to worm attacks. Thus, the rapid spread of computer worms and viruses has become a major threat to our normal life. The success of this project and subsequent implementation of the proposed procedures may be the first steps toward reducing or limiting damage caused by internet worm attacks to any organization using a computer network of sufficient size.

By using evolution as a tool to create better and safer networks, this project aims to improve the current state of network design. By doing so, it may be possible to prevent or reduce damage associated with worm infections on computer networks. In addition, the techniques used for optimizing security in a network may be easily adopted for use for optimizing other characteristics of networks, such as speed, cost, and stability.

We are still at a very preliminary stage. Much more simulations on larger scale networks need to be done and more rigorous analyses are needed to fully understand the significance of our findings. Our model needs to take more factors into consideration so that it becomes more realistic and results can be implemented in real applications.
2. Models and Methods

2.1 Definitions and Parameters

In order to adequately describe our model and our simulation process, it is necessary to define several terms, for the sake of precision. Although a network could be viewed as a graph as defined in graph theory, the objective of our project is to develop a procedure applicable to realistic computer networks, not abstract mathematical models of networks.

**Definition 2.1:** A node in a computer network is defined as a point on the network, which in real-world applications may represent anything from a client PC to a supercomputing cluster.

**Definition 2.2:** A connection between two nodes is defined as a line connecting the nodes together so that information (such as a computer worm) can travel from one node to another following the line. The two nodes are said to be adjacent to each other if they are connected directly by one connection.

Travel time needed to go from one node to its adjacent node is assumed to be constant for all connections in our model. This is to keep the model simple. It is also because the spread of a worm is normally not limited by physical distance or travel time.

**Definition 2.3:** Two nodes are said to be linked if one can travel from one node to another through one or more connections linking the nodes together. The linked connection set is called a path between the linked nodes.

**Definition 2.4:** A computer network is defined as a set of nodes linked by a set of connections and there is at least one path between any given two nodes in the network (connectivity). A large-scale network is a network with between 100-500 nodes.

The connectivity requirement is to guarantee that every node on the network is able to be infected, and thus eliminating the effect of isolated nodes on the results. A text file can be used to specify a network completely by defining nodes and each node’s adjacent nodes. Fig. 2.1 gives some examples of simple networks.
Definition 2.5: A computer worm is a computer program that spreads from infected nodes to other adjacent nodes across a network, in turn infecting those nodes. The spread of a worm is defined as the collective transmission of a worm across a network.

The spread of a worm may be affected by many factors including network design, anti-virus software installed on the computers and network, computer operating systems, user-dependent factors, etc. We limit our investigation on network structure only. The effects of various other parameters may by topics of further extension in future research.

Definition 2.6 A global Internet in this project is defined as a network with 360,000 nodes which is randomly generated by a simulation package SSFNet and used as the source of worm infection during the SSFNet simulation.

Fig. 2.1 Examples of basic subnets and initial Input networks.
In order to foster positive change in the evolution, it is necessary to introduce a measure of the fitness of each network in the form of an objective function. To this end, we have:

**Definition 2.7** For an infected network, its *infection rate* at time \( t \) is defined as:

\[
R_{\text{infect}} (t) = \frac{\text{Infected Nodes at time } t}{\text{Total Number of Nodes}}. \quad (1)
\]

The *averaged infection rate* of an infected network from time step 1 to time step \( N \) is defined as:

\[
R_{\text{ave}} = \frac{R_{\text{infect}} (t_1) + R_{\text{infect}} (t_2) + \cdots + R_{\text{infect}} (t_N)}{N}. \quad (2)
\]

where \( N \) is the total time steps of the simulated period.

Both \( R_{\text{infect}} \) and \( R_{\text{ave}} \) can be used to measure infection level of an infected network, with \( R_{\text{ave}} \) as the preferred measure because it contains averaged information over a period of time. Choosing \( R_{\text{ave}} \) as our objective function, then the network optimization problem becomes the problem of minimizing \( R_{\text{ave}} \).

\( R_{\text{ave}} \) values are used to compare networks objectively for each generation. By comparing the value of the objective function for both a conventional network and the evolved network, we can determine the degree of improvement for the entire evolution. To calculate the value of the objective function efficiently, a separate program was written to parse the output from the simulation.

During the evolution, we need to measure and control a number of different parameters and values in order to measure the progress of the evolution versus an independent variable. In this project, we concentrate on the structure of the network being tested. In order to compare structures, however, it is necessary to define some objective measures of specific properties of a network’s structure. Specifically, we would like to measure centralization, hierarchy, distribution of node types, and the
number of connections in a network. These values will all vary as the network is randomly mutated across generations. The most important measure of network connectivity for the purposes of worm spread is the distribution of connections across a network. We use the standard deviation of the number of connections for each node (the order of the node) for a given network as a rough measure of network centralization and distribution features.
2.2 Model Design Goals

For the final product, i.e. the optimized network structure, certain goals are defined for its security and performance. First, and perhaps most importantly, the final network must be significantly more secure than the original input conventional network. Quantitatively, this means that the final network must be characterized by a significant decrease in the number of infected nodes during a worm infection. This will be calculated with the NWS worm simulation. Secondly, the network cannot be significantly slower than a conventional network. Thirdly, the network cannot be noticeably less reliable than the original network. For example, a theoretical linear network with all nodes connected along a line will be the most secure because it takes the longest time to be fully infected. However, this network will be the slowest of all possible networks and the least reliable (if one node malfunctions, the entire network goes down). Because of this, it is necessary to maintain reasonable levels for both speed and reliability. This is accomplished by maintaining relatively constant node orders in the mutation and combination algorithms. However, a rigorous method for achieving this is beyond the scope of this project.

The main goals of this program design are centered around the improvement of network design with regard to worm infection. These goals are:

1. Create a more secure network, that is, one that has a smaller percentage of infection.
2. Improve upon existing network design strategies.
3. Develop a set of equations governing the infection of networks based on certain parameters involving the network’s structure.
2.3 Program Structure

Evolution is a process by which all species develop from earlier forms of life. On this theory, natural variation in the genetic material of a population favors reproduction by some individuals more than others, so that over the generations all members of the population come to possess the favorable or better forms. The main component of any evolution algorithm is the introduction of positive variation into the system being optimized. Variation is the raw material necessary for possible improvement of each generation. Thus, for this project, it was necessary to create several means of introducing variations, namely combination and mutation.

Definition 2.8: Combination is a process which takes two network configurations and generates a network that possesses the characteristics of both of the input networks.

Definition 2.9: Mutation is a process that takes only one network as input and randomly changes the connections between nodes, while preserving the connectivity of the network, and outputs the resulting network.

In order to convert network variation into actual positive change, however, we introduce a third mechanism: selection.

Definition 2.10: Selection is a process which chooses the two networks from each generation with the lowest infection values (Rave values), and passes them to the next generation.

Using these three principles, we create a system that is capable of improving and optimizing network structures.
Fig. 2.2 gives a diagram of our program design. In order to effectively simulate the propagation of a worm across a computer network of the target size, this project used the Network Worm Simulator Perl package. This package served as a tool to measure the fitness of each network by infecting a given network with a specified worm. NWS was used only as a measuring tool, however, and the actual evolution and optimization of the network structures was done by separate Perl programs. The output from the simulation was parsed by a Perl program into a readable format and then used as input to generate that network’s fitness level. For each generation of networks, the infection levels were compared, and the two lowest and their respective networks were chosen to “seed” the next generation. These two networks were combined to form a single network that was then varied randomly to populate the following generation. This process was repeated
over several generations, until the resultant network structure showed significant signs of improvement over conventional networks and satisfied the original design goals.

A set of Perl programs was written and used primarily to connect the various simulations and their results, and to control the evolution of the networks. The reason for implementing the programs in Perl lies mainly in Perl’s text manipulation capabilities. When mutating a network, a program must scan through an input configuration and make intelligent changes, while preserving the structure and meaning of the network configuration file. In addition, parsing the output of the simulation into a useable format also requires a language with flexible and powerful pattern recognition features. This, combined with Perl’s rapid development cycle and portability, led to its use as the programming language of choice for the implementation of the programs.

A separate package SSFNet was used to simulate network infection under the condition that the local network is connected to the global Internet and infection is propagated through internet ISP addresses and to verify the results of the evolution. SSFNet is implemented in Java and is based on random worm spread patterns as opposed to the simple spread patterns used by NWS. Additionally, coupling the global Internet with the local network being tested, as SSFNet does, to investigate worm spread and network infection leads to more realistic models and is a potential area for future projects.
3. Results and Analysis

Computational simulations were conducted using NWS on 1000 randomly generated networks to test the effectiveness of our evolution selection method. SSFNet was then used to test network infection under global network environment and validate our method and outcome.

3.1 Results with Network Worm Simulator

In order to compare the effects of network structure on worm spread, the worm simulation was run on approximately 1000 randomly generated networks. These networks were chosen to have a reasonable coverage of basic network types, and then varied by random mutating and then combining, using the methods outlined above, while keeping the number of nodes constant. An optimized network after 100 evolutionary generations obtained from an initial input two-layer star-shaped network (see Fig. 2.1) is given by Appendix A. The average infection rate for each generation is given by Fig, 3.1. The original seed network showed an average infection rate of about 80%. After running through 100 generations with natural evolutionary selections of 10 networks for each generation, the final network had reduced $R_{ave}$ value to 47%, a significant decrease in infection. By analyzing the progress of the network’s evolution through 100 generations, it was observed that the average infection levels for the network appears to be inversely proportional to the generation. The curve fitting the numerical data is given by:

$$R_{ave} = (1032*(G + 27.6) + 39.93) \%, \quad (3)$$

which was obtained by the least-squares approximation method.
Fig. 3.1 Natural evolutionary selection considerably reduces network infection rate.

To determine the characteristics of the optimized networks which led to their increased security, the average infection levels for all 1000 networks tested were plotted against the connection distribution value which is defined as the standard deviation of node orders (see Fig. 3.2). Node order is defined as the number of connections each node has. As this connection distribution value becomes larger, the number of connections per node for the network becomes more varied, leading to a higher level of centralization. If this value is relatively small, however, it indicates that the connections are uniformly distributed, leading to a decentralized network.
Fig. 3.2 Network infection rate shows correlation with standard deviation of node orders.

Although the data was relatively unordered, we observed a general trend toward higher infection levels as the connections became less spread out. The data was modeled linearly with an equation generated by a least squares regression, shown here:

\[ R_{\text{ave}} = (26.24 \times D - 1.425)\% \]

where D is the standard deviation of node orders. These results verify the original hypothesis: that a distributed network leads to a more secure network and a reduction in worm infection levels. In addition, the relationship between the average
infection rates and evolution generations also demonstrates the effectiveness of our evolution process as a useful tool for network design improvement.
3.2 Validation Using SSFNet

Validation is a necessary step for computational simulation research before results can be implemented in real-life applications. A second package known as SSFNet, written in Java, was used to verify the results obtained through the NWS simulations. SSFNet is a much more complex and realistic package than NWS, and uses random worm spread patterns to accurately model the spread of a worm across the input network under a global Internet environment. Some program running parameters are given as follows:

- Local network connected to a global Internet made of 360000 nodes;
- Simulated time for each case of 56700 seconds.
- Time resolution: 1000000000 ticks / seconds.
- Update interval: 60 seconds (output infected nodes by that time).

SSFNet is much more computationally intensive than NWS, and therefore was not practical for use in the evolutionary system. Fig. 3.3 (red-line curve) shows the infection rate of the initial two-layer star-shaped network (see Fig. 2.1) vs. time as given by SSFNet. We see that the network quickly reaches nearly 100% infection level, and was therefore extremely vulnerable to worm infection. Fig. 3.3 (blue-line curve) gives the infection rate of the optimized network. The optimized network consists of a much more varied and evenly spread topology than the star network. Instead of quickly becoming fully infected, this network slows the infection as it reaches about 50% infection level, a much more desirable outcome than 100% infection. Other examples have shown similar improvements. Using SSFNet as a validation tool, we were able to see that networks
optimized by the evolutionary procedure have much better infection rate. Experiments using real networks are needed to validate our computational predictions.

![Graph showing infection rate over time]

**Fig. 3.3** Simulation using SSFNet shows that optimized network has much lower infection rate compared to the original input network. Red-line: the two-layer star-shaped network reaches nearly 100% infection level within 56700 seconds; Blue-line: the optimized network slows the infection as it reaches about 50% infection level, a much more desirable outcome than 100% infection.
4. Discussion

For the future, this project may be extended into a full-fledged network development application. For example, a company requires a network of certain specifications, and enters those values into the program. The program will then run the evolution, and return an optimized solution to the company’s needs. Of course, this approach has many limitations. Much work is needed to improve the model and procedure. Often, evolution within a hypothetical system may generate either meaningless or impractical solutions to a problem, such as when the output network has far too many connections to be practical, or is laid out in a way that is physically impossible.

Another possible extension would be to conduct the simulation on more realistic node types, perhaps even with individual specifications. This might, however, prove to be too computationally demanding to be feasible. It would also be of use to run the simulation using various types of worms, or simply various types of Internet-borne threats to security, such as spam, Trojan horses, denial of service attacks, etc. Doing so would add far more real-world value to the simulation’s results than if it were run using only one type of attack or worm. This type of extension would be relatively simple, given the extensible nature of the NWS package.
5. Conclusions

The standard design strategies for computer networks often lead to networks that are vulnerable to worm infection. Therefore, to create more secure networks, an evolutionary approach was taken in order to optimize network structures for security, and was tested with 1000 networks. The optimized networks show an average of 50% improvement in average infection rate (from about 80% infection rate to just below 50% infection rate). Infection rate is inversely proportional to evolution generation for cases tested, and a relationship between the structure of a network and its vulnerability to infection was also found. Optimized networks were shown to be more decentralized and more distributed when compared to a traditional network. Some validation of our results was given by SSFNet under a global Internet environment.

Acknowledgement

I would like to thank all of the people involved in this project for their aid and assistance, specifically Ms. Lang for her guidance and advice, Prof. Meng for helping me with my programs, Prof. Liljenstam for providing me with the S.O.S. Perl package, and all of the people involved with the development of NWS and SSFNet, without which this project would have been considerably more difficult.
References


## Appendix A: Nodes and Connections of an Optimized Network

This table contains the nodes and their connections of the network optimized by the evolution procedure from the two-layer star-shaped network.

<table>
<thead>
<tr>
<th>Node</th>
<th>Adjacent Nodes</th>
<th>Node</th>
<th>Adjacent Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2, 4, 24, 35, 46, 57, 68, 79, 90, 9</td>
<td>2</td>
<td>1, 3, 4, 5, 6, 9, 10, 11, 12</td>
</tr>
<tr>
<td>3</td>
<td>2, 45</td>
<td>4</td>
<td>2, 56, 62, 47</td>
</tr>
<tr>
<td>5</td>
<td>2, 18</td>
<td>6</td>
<td>2, 28, 92, 53</td>
</tr>
<tr>
<td>7</td>
<td>81, 46, 56</td>
<td>8</td>
<td>40, 102, 22</td>
</tr>
<tr>
<td>9</td>
<td>2, 1, 77</td>
<td>10</td>
<td>2, 108</td>
</tr>
<tr>
<td>11</td>
<td>2, 32</td>
<td>12</td>
<td>2, 91</td>
</tr>
<tr>
<td>13</td>
<td>14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 50</td>
<td>14</td>
<td>13, 47</td>
</tr>
<tr>
<td>15</td>
<td>13, 100, 25, 32</td>
<td>16</td>
<td>13, 57, 31, 38</td>
</tr>
<tr>
<td>17</td>
<td>13, 74, 83</td>
<td>18</td>
<td>13, 5, 59</td>
</tr>
<tr>
<td>19</td>
<td>13, 105, 38, 41</td>
<td>20</td>
<td>13, 30</td>
</tr>
<tr>
<td>21</td>
<td>13, 31, 33</td>
<td>22</td>
<td>13, 55, 8</td>
</tr>
<tr>
<td>23</td>
<td>13, 13, 24, 105</td>
<td>24</td>
<td>1, 25, 26, 27, 28, 29, 30, 31, 33, 34</td>
</tr>
<tr>
<td>25</td>
<td>24, 15, 72, 105</td>
<td>26</td>
<td>24, 42, 29</td>
</tr>
<tr>
<td>27</td>
<td>24, 51, 85</td>
<td>28</td>
<td>24, 6, 96</td>
</tr>
<tr>
<td>29</td>
<td>24, 26, 98</td>
<td>30</td>
<td>24, 20</td>
</tr>
<tr>
<td>31</td>
<td>24, 47, 21, 16</td>
<td>32</td>
<td>11, 15, 34</td>
</tr>
<tr>
<td>33</td>
<td>24, 97, 60, 21, 78</td>
<td>34</td>
<td>24, 60, 32</td>
</tr>
<tr>
<td>35</td>
<td>1, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 88</td>
<td>36</td>
<td>35, 91</td>
</tr>
<tr>
<td>37</td>
<td>35</td>
<td>38</td>
<td>35, 19, 16</td>
</tr>
<tr>
<td>39</td>
<td>35, 84</td>
<td>40</td>
<td>35, 8, 109</td>
</tr>
<tr>
<td>41</td>
<td>35, 44, 19</td>
<td>42</td>
<td>35, 26, 58</td>
</tr>
<tr>
<td>43</td>
<td>35</td>
<td>44</td>
<td>35, 41, 46</td>
</tr>
<tr>
<td>45</td>
<td>35, 3</td>
<td>46</td>
<td>1, 47, 48, 49, 50, 52, 53, 54, 55, 56, 57, 44</td>
</tr>
<tr>
<td>47</td>
<td>46, 31, 14, 4</td>
<td>48</td>
<td>46</td>
</tr>
<tr>
<td>49</td>
<td>46, 63</td>
<td>50</td>
<td>46, 13, 86, 64, 100</td>
</tr>
<tr>
<td>51</td>
<td>27, 108, 54</td>
<td>52</td>
<td>46, 60, 74</td>
</tr>
<tr>
<td>53</td>
<td>46, 6</td>
<td>54</td>
<td>46, 51, 57</td>
</tr>
<tr>
<td>55</td>
<td>46, 22</td>
<td>56</td>
<td>46, 4, 7</td>
</tr>
<tr>
<td>57</td>
<td>1, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 16, 54, 108</td>
<td>58</td>
<td>57, 42, 88</td>
</tr>
<tr>
<td>59</td>
<td>57, 18, 77</td>
<td>60</td>
<td>57, 52, 34, 33, 65</td>
</tr>
<tr>
<td>61</td>
<td>57, 68</td>
<td>62</td>
<td>57, 4</td>
</tr>
<tr>
<td>63</td>
<td>57, 49</td>
<td>64</td>
<td>57, 85, 91, 50</td>
</tr>
<tr>
<td>65</td>
<td>57, 60</td>
<td>66</td>
<td>57</td>
</tr>
<tr>
<td>67</td>
<td>57, 78</td>
<td>68</td>
<td>1, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 81</td>
</tr>
<tr>
<td>69</td>
<td>68</td>
<td>70</td>
<td>68, 77</td>
</tr>
<tr>
<td>71</td>
<td>68, 75, 78, 99, 96</td>
<td>72</td>
<td>68, 25</td>
</tr>
<tr>
<td>73</td>
<td>68</td>
<td>74</td>
<td>68, 17, 52</td>
</tr>
<tr>
<td>75</td>
<td>68, 71</td>
<td>76</td>
<td>68, 97, 93</td>
</tr>
<tr>
<td>77</td>
<td>68, 70, 59, 9</td>
<td>78</td>
<td>68, 71, 67, 33</td>
</tr>
<tr>
<td>79</td>
<td>1, 80, 81, 82, 83, 84, 85, 87, 88, 89</td>
<td>80</td>
<td>79</td>
</tr>
<tr>
<td>81</td>
<td>79, 7, 86</td>
<td>82</td>
<td>79, 88, 108</td>
</tr>
<tr>
<td>83</td>
<td>79, 17</td>
<td>84</td>
<td>79, 39</td>
</tr>
<tr>
<td>85</td>
<td>79, 64, 27</td>
<td>86</td>
<td>81, 50</td>
</tr>
<tr>
<td>87</td>
<td>79, 95</td>
<td>88</td>
<td>79, 93, 82, 58, 35</td>
</tr>
<tr>
<td>89</td>
<td>79</td>
<td>90</td>
<td>1, 92, 94, 95, 96, 98, 99, 100</td>
</tr>
<tr>
<td>91</td>
<td>12, 36, 64</td>
<td>92</td>
<td>90, 6</td>
</tr>
<tr>
<td>93</td>
<td>88, 76, 108</td>
<td>94</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>90.87</td>
<td></td>
<td>90,28,71</td>
</tr>
<tr>
<td>---</td>
<td>------</td>
<td>---</td>
<td>---------</td>
</tr>
<tr>
<td>95</td>
<td></td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>33,76</td>
<td></td>
<td>90,29</td>
</tr>
<tr>
<td>99</td>
<td>90,71</td>
<td>100</td>
<td>90,15,50</td>
</tr>
<tr>
<td>101</td>
<td>102,103,104,106,107,109,110,111</td>
<td>102</td>
<td>101,8</td>
</tr>
<tr>
<td>103</td>
<td>101</td>
<td>104</td>
<td>101</td>
</tr>
<tr>
<td>105</td>
<td>19,25</td>
<td>106</td>
<td>101</td>
</tr>
<tr>
<td>107</td>
<td>101,111</td>
<td>108</td>
<td>51,10,82,93,57</td>
</tr>
<tr>
<td>109</td>
<td>101,40</td>
<td>110</td>
<td>101</td>
</tr>
<tr>
<td>111</td>
<td>101,107</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix B: Selected Programs

B.1 Main Program Source Code

#!/usr/bin/perl
#
# Main program for simulation
#

%fitnessValues;
my $defaultNetwork = $ARGV[0];
my $numberOfGenerations = $ARGV[1];
my $id = '1';
my $gen = '1';

$; = ‘~’;

open(NETSTATS, “>table.txt”);
open(GENS, “>generations.txt”);

`perl generic.pl $defaultNetwork > output$gen~$id.out`;
$fitnessValues{$gen,$id} = `perl evaluator.pl output$gen~$id.out`;

print "Network $id: ";
print $fitnessValues{$gen,$id}, "n"

print NETSTATS `perl descriptor.pl $defaultNetwork`;
my $initialInfection = $fitnessValues{$gen,$id};
print NETSTATS ",", $fitnessValues{$gen,$id}, "n";

for $run (2..10)
{
    $id++;
    `perl mutator.pl $defaultNetwork mutated$gen~$id.txt`;
    `perl generic.pl mutated$gen~$id.txt > output$gen~$id.out`;
    $fitnessValues{$gen,$id} = `perl evaluator.pl output$gen~$id.out`;
    print "Network $id: ";
    print $fitnessValues{$gen,$id}, "n"

    print NETSTATS `perl descriptor.pl mutated$gen~$id.txt`;
    print NETSTATS ",", $fitnessValues{$gen,$id}, "n";
}

@keyArray = reverse(sort {$fitnessValues{$b} <=> $fitnessValues{$a}} keys %fitnessValues);
$bestKey = $keyArray[0];
print $fitnessValues{$bestKey}, "n"

for $counter (1..$numberOfGenerations - 1)
{
    $defaultNetwork = "mutated$bestKey.txt";
    $gen++;
    $id = '1';

    `copy mutated$bestKey.txt mutated$gen~1.txt`;

    "~";
open(GENS, ">generations.txt");

    `perl generic.pl $defaultNetwork > output$gen~$id.out`;
    $fitnessValues{$gen,$id} = `perl evaluator.pl output$gen~$id.out`;

    print "Network $id: ";
    print $fitnessValues{$gen,$id}, "n"

    print NETSTATS `perl descriptor.pl $defaultNetwork`;
    my $initialInfection = $fitnessValues{$gen,$id};
    print NETSTATS ",", $fitnessValues{$gen,$id}, "n";

    for $run (2..10)
    {
        $id++;
        `perl mutator.pl $defaultNetwork mutated$gen~$id.txt`;
        `perl generic.pl mutated$gen~$id.txt > output$gen~$id.out`;
        $fitnessValues{$gen,$id} = `perl evaluator.pl output$gen~$id.out`;
        print "Network $id: ";
        print $fitnessValues{$gen,$id}, "n"

        print NETSTATS `perl descriptor.pl mutated$gen~$id.txt`;
        print NETSTATS ",", $fitnessValues{$gen,$id}, "n";
    }

    @keyArray = reverse(sort {$fitnessValues{$b} <=> $fitnessValues{$a}} keys %fitnessValues);
    $bestKey = $keyArray[0];
    print $fitnessValues{$bestKey}, "n"

    for $counter (1..$numberOfGenerations - 1)
    {
        $defaultNetwork = "mutated$bestKey.txt";
        $gen++;
        $id = '1';

        `copy mutated$bestKey.txt mutated$gen~1.txt`;

        "~";
open(GENS, ">generations.txt");

    `perl generic.pl $defaultNetwork > output$gen~$id.out`;
    $fitnessValues{$gen,$id} = `perl evaluator.pl output$gen~$id.out`;

    print "Network $id: ";
    print $fitnessValues{$gen,$id}, "n"

    print NETSTATS `perl descriptor.pl $defaultNetwork`;
    my $initialInfection = $fitnessValues{$gen,$id};
    print NETSTATS ",", $fitnessValues{$gen,$id}, "n";

    for $run (2..10)
    {
        $id++;
        `perl mutator.pl $defaultNetwork mutated$gen~$id.txt`;
        `perl generic.pl mutated$gen~$id.txt > output$gen~$id.out`;
        $fitnessValues{$gen,$id} = `perl evaluator.pl output$gen~$id.out`;
        print "Network $id: ";
        print $fitnessValues{$gen,$id}, "n"

        print NETSTATS `perl descriptor.pl mutated$gen~$id.txt`;
        print NETSTATS ",", $fitnessValues{$gen,$id}, "n";
    }

    @keyArray = reverse(sort {$fitnessValues{$b} <=> $fitnessValues{$a}} keys %fitnessValues);
    $bestKey = $keyArray[0];
    print $fitnessValues{$bestKey}, "n"

    for $counter (1..$numberOfGenerations - 1)
    {
        $defaultNetwork = "mutated$bestKey.txt";
        $gen++;
        $id = '1';

        `copy mutated$bestKey.txt mutated$gen~1.txt`;

        "~";
open(GENS, ">generations.txt");

    `perl generic.pl $defaultNetwork > output$gen~$id.out`;
    $fitnessValues{$gen,$id} = `perl evaluator.pl output$gen~$id.out`;

    print "Network $id: ";
    print $fitnessValues{$gen,$id}, "n"

    print NETSTATS `perl descriptor.pl $defaultNetwork`;
    my $initialInfection = $fitnessValues{$gen,$id};
    print NETSTATS ",", $fitnessValues{$gen,$id}, "n";

    for $run (2..10)
    {
        $id++;
        `perl mutator.pl $defaultNetwork mutated$gen~$id.txt`;
        `perl generic.pl mutated$gen~$id.txt > output$gen~$id.out`;
        $fitnessValues{$gen,$id} = `perl evaluator.pl output$gen~$id.out`;
        print "Network $id: ";
        print $fitnessValues{$gen,$id}, "n"

        print NETSTATS `perl descriptor.pl mutated$gen~$id.txt`;
        print NETSTATS ",", $fitnessValues{$gen,$id}, "n";
    }

    @keyArray = reverse(sort {$fitnessValues{$b} <=> $fitnessValues{$a}} keys %fitnessValues);
    $bestKey = $keyArray[0];
    print $fitnessValues{$bestKey}, "n"

    for $counter (1..$numberOfGenerations - 1)
    {
        $defaultNetwork = "mutated$bestKey.txt";
        $gen++;
        $id = '1';

        `copy mutated$bestKey.txt mutated$gen~1.txt`;

        "~";
open(GENS, ">generations.txt");

    `perl generic.pl $defaultNetwork > output$gen~$id.out`;
    $fitnessValues{$gen,$id} = `perl evaluator.pl output$gen~$id.out`;

    print "Network $id: ";
    print $fitnessValues{$gen,$id}, "n"

    print NETSTATS `perl descriptor.pl $defaultNetwork`;
    my $initialInfection = $fitnessValues{$gen,$id};
    print NETSTATS ",", $fitnessValues{$gen,$id}, "n";

    for $run (2..10)
    {
        $id++;
        `perl mutator.pl $defaultNetwork mutated$gen~$id.txt`;
        `perl generic.pl mutated$gen~$id.txt > output$gen~$id.out`;
        $fitnessValues{$gen,$id} = `perl evaluator.pl output$gen~$id.out`;
        print "Network $id: ";
        print $fitnessValues{$gen,$id}, "n"

        print NETSTATS `perl descriptor.pl mutated$gen~$id.txt`;
        print NETSTATS ",", $fitnessValues{$gen,$id}, "n";
    }

    @keyArray = reverse(sort {$fitnessValues{$b} <=> $fitnessValues{$a}} keys %fitnessValues);
    $bestKey = $keyArray[0];
    print $fitnessValues{$bestKey}, "n"

    for $counter (1..$numberOfGenerations - 1)
    {
        $defaultNetwork = "mutated$bestKey.txt";
        $gen++;
        $id = '1';

        `copy mutated$bestKey.txt mutated$gen~1.txt`;

        "~";
open(GENS, ">generations.txt");

    `perl generic.pl $defaultNetwork > output$gen~$id.out`;
    $fitnessValues{$gen,$id} = `perl evaluator.pl output$gen~$id.out`;

    print "Network $id: ";
    print $fitnessValues{$gen,$id}, "n"

    print NETSTATS `perl descriptor.pl $defaultNetwork`;
    my $initialInfection = $fitnessValues{$gen,$id};
    print NETSTATS ",", $fitnessValues{$gen,$id}, "n";

    for $run (2..10)
    {
        $id++;
        `perl mutator.pl $defaultNetwork mutated$gen~$id.txt`;
        `perl generic.pl mutated$gen~$id.txt > output$gen~$id.out`;
        $fitnessValues{$gen,$id} = `perl evaluator.pl output$gen~$id.out`;
        print "Network $id: ";
        print $fitnessValues{$gen,$id}, "n"

        print NETSTATS `perl descriptor.pl mutated$gen~$id.txt`;
        print NETSTATS ",", $fitnessValues{$gen,$id}, "n";
    }

    @keyArray = reverse(sort {$fitnessValues{$b} <=> $fitnessValues{$a}} keys %fitnessValues);
    $bestKey = $keyArray[0];
    print $fitnessValues{$bestKey}, "n"

    for $counter (1..$numberOfGenerations - 1)
    {
        $defaultNetwork = "mutated$bestKey.txt";
        $gen++;
        $id = '1';

        `copy mutated$bestKey.txt mutated$gen~1.txt`;

        "~";
open(GENS, ">generations.txt");

    `perl generic.pl $defaultNetwork > output$gen~$id.out`;
    $fitnessValues{$gen,$id} = `perl evaluator.pl output$gen~$id.out`;

    print "Network $id: ";
    print $fitnessValues{$gen,$id}, "n"

    print NETSTATS `perl descriptor.pl $defaultNetwork`;
    my $initialInfection = $fitnessValues{$gen,$id};
    print NETSTATS ",", $fitnessValues{$gen,$id}, "n";

    for $run (2..10)
    {
        $id++;
        `perl mutator.pl $defaultNetwork mutated$gen~$id.txt`;
        `perl generic.pl mutated$gen~$id.txt > output$gen~$id.out`;
        $fitnessValues{$gen,$id} = `perl evaluator.pl output$gen~$id.out`;
        print "Network $id: ";
        print $fitnessValues{$gen,$id}, "n"

        print NETSTATS `perl descriptor.pl mutated$gen~$id.txt`;
        print NETSTATS ",", $fitnessValues{$gen,$id}, "n";
    }
print "$defaultNetwork
";

`perl generic.pl $defaultNetwork > output.out`
$fitnessValues{$gen,$sid} = `perl evaluator.pl output.out`

print "Network $id: ";
print $fitnessValues{$gen,$id}, "
"

print NETSTATS `perl descriptor.pl $defaultNetwork`
print NETSTATS ",", $fitnessValues{$id}, "\n"

for $run (2..10)
{
    `perl mutator.pl $defaultNetwork mutated$gen~$run.txt`
    `perl generic.pl mutated$gen~$run.txt > output$gen~$run.out`
    $fitnessValues{$gen,$run} = `perl evaluator.pl output$gen~$run.out`
    print "Network $run: ";
    print $fitnessValues{$gen,$run}, "\n"
    print NETSTATS `perl descriptor.pl mutated$gen~$run.txt`
    print NETSTATS ",", $fitnessValues{$gen,$run}, "\n"
}

@keyArray = reverse(sort {$fitnessValues{$b} <=> $fitnessValues{$a}} keys %fitnessValues);
$bestKey = $keyArray[0];
print "$bestKey: ";
print $fitnessValues{$bestKey}, "\n"
print GENS $fitnessValues{$bestKey}, "\n"

$finalInfection = $fitnessValues{$bestKey};

print "\n"
print "Initial Network: $initialInfection"
print "Final Network: $finalInfection"

close NETSTATS, GENS;
B.2 Mutator Program Source Code

#!/C:\Perl\bin -w
#
# Randomly mutate current network configuration
#

if ($ARGV[0] eq $ARGV[1])
{
    exit 0;
}

sub generateAdjacencyMatrix
{
    my @adjacencies;
    while (<>)
    {
        chop;
        push @adjacencies, [split ",",$_];
    }

    return @adjacencies;
}

sub generateNodeList
{
    my @inputMatrix = @_; 
    my @outputList;

    foreach $adjRow (@inputMatrix)
    {
        push @outputList, $adjRow->[0];
    }

    return @outputList;
}

sub isIn
{
    my @aList = @_; 
    my $value = shift @aList;

    foreach $element (@aList)
    {
        return 1 if $element == $value;
    }

    return 0;
}

sub inverseNodes (@@)
{
    my $inputList = shift;
    my $universalList = shift;

    my @output;
foreach $element (@$universalList)

    {push @output, $element if !(isIn($element, @$inputList));}

return @output;

}

sub removeNode

{my @nodeList = @_; return @nodeList if $#nodeList == 1;

    my $randIndex = rand($#nodeList) + 1;
    splice(@nodeList, $randIndex, 1);

    return @nodeList;
}

sub addNode (@@@)

{my $nodeList = shift;
    my $universalList = shift;
    my @output = @$nodeList;
    my @inverseArray = inverseNodes(@$nodeList, @$universalList);

    my $randIndex = rand($#inverseArray + 1);
    push @output, $inverseArray[$randIndex];

    return @output;
}

sub swapNode

{my $nodeListA = shift;
    my $nodeListB = shift;
}

sub printMatrix (@@)

{my $fh = shift;
    my @inputArray = @_;
    foreach my $row (@inputArray)
    {
        print $fh join("", @$row), "n";
    }
}

open(OUTPUT, ">$ARGV[1]");

my @adjacencyMatrix = generateAdjacencyMatrix;
my @nodeList = generateNodeList @adjacencyMatrix;

foreach my $adjRow (@adjacencyMatrix)
{ if (rand(4) <= 1)
  { $adjRow = [removeNode(@$adjRow)]; }
}

if (rand(4) <= 1)
{ $adjRow = [addNode(@$adjRow, @nodeList)]; }

printMatrix OUTPUT, @adjacencyMatrix;
close OUTPUT;
B.3 Evaluator Program Source Code

```perl
#!/C:/Perl/bin
#
# Extract data and calculate fitness
#

sub total
{
    my $sum = 0;
    foreach $something (@_)
    {
        $sum = $sum + $something;
    }
    return $sum;
}

$fitnessindex = 0;
$currentinfection = 0;
@infections;
@infectiontimes;

$netsize;

while (<>)
{
    if(/Total entities/) 
    {
        my ($size) = /population: ('d+)/;
        $netsize = $size;
    }
    if (/((d+)(d+)(d+))/) 
    {
        push (@infections, $3);
    }
}

while (@infections[0] == 0)
{
    shift (@infections);
}

$fitnessindex = (total(@infections) / ($#infections + 1)) * 100 / $netsize;

open(TABLE, ">> values.txt") or die "Can't open file: $!";

print TABLE $fitnessindex, ",";
print $fitnessindex;

close (TABLE);
```
B.4 Simulator Program Source Code

#!/usr/bin/perl
# Copyright 2003, Bruce Ediger
# Modifications by Mason Tang
# This file is part of NWS.
#
# NWS is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
# NWS is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with NWS; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
#
# $Id: generic.pl,v 1.10 2003/09/20 05:02:58 bediger Exp $

use strict;
use diagnostics;

use vars qw($opt_S $opt_A $opt_V $opt_w $opt_s $opt_x $opt_X $opt_h $opt_B);
use Getopt::Std;
use NWS::Network;
use NWS::Message;
use NWS::Host;
use NWS::Software;

sub getAdjList
{
  my @adjList = @_; 
  my $node = shift @adjList;

  foreach my $aList (@adjList)
  {
    if (@$aList[0] == $node)
    {
      my @output = @$aList;
      shift @output;
      return @output;
    }
  }
}

# Parse input configuration into adjacency matrix

my @adjacencies;
while (<>)
{
  chop;
  push @adjacencies, [split ",",$_];
}
my ($seed, $address_space, $victim_cnt, $worm_cnt, $stop_proportion, $banded) = (0, 65536, 110, 1, 0.01, 0);

gopt('S:A:V:B:w:s:');

$seed = $opt_S if $opt_S;
$address_space = $opt_A if $opt_A;
$victim_cnt = $opt_V if $opt_V;
$worm_cnt = $opt_w if $opt_w;
$stop_proportion = $opt_s if $opt_s;
$banded = $opt_B if $opt_B;
&usage if $opt_x or $opt_X or $opt_h;

# Proportion of address space that actually has a Host
my $fullness = ($victim_cnt + $worm_cnt)/$address_space;

my $network = new Network $address_space, $seed;

my $Aworm_code = q{
  my ($host, $software, $phase, $msg) = @_;  
  if ($phase == $Host::Run) 
    
  $host->SendMsg(
    new Message
    $adjNode,
    "OS-CPU",
    "Bvictim",
    $software->{function});
  
  elsif ($phase == $Host::Init2) {
    $software->{true_name} = 'Aworm';
    $software->{exploitable} = 0; # Aworm can't re-exploit this machine
  }
};

for (my $i = 1; $i <= $victim_cnt; ++$i) { 
  my $host2 = new Host 'OS-CPU', $i, [getAdjList($i, @adjacencies)];
  my $sw3 = new Software 'Bvictim', '', 1; # no code, exploitable
  $host2->AddSoftware($sw3);
  my $addr = $network->AddHost($host2);
}

for (my $i = $victim_cnt + 1; $i <= $worm_cnt + $victim_cnt; ++$i) { 
  my $host = new Host 'OS-CPU', $i, [getAdjList($i, @adjacencies)];
  my $software = new Software 'Aworm', $Aworm_code, 0;
  $host->AddSoftware($software);
  my $addr = $network->AddHost($host);
}
my $steps = 20;
my $delta = 1;
print "Running a Network through $steps time steps\n";

print "# Generic SIS model with no disinfection.\n";
print "# Address space size:     $address_space\n";
if ($banded) {
   print "# Address allocated in $banded bands to victim hosts\n";
} else {
   print "# Random address allocation to hosts\n";
}
print "# Starting victim count:    $victim_cnt\n";
print "# Starting worm count:     $worm_cnt\n";
my $start_timestamp = time;
$network->PrintCounts;
for (my $i = 1; $i < $steps; $i += $delta) {
   $network->Run($i, $delta)->PrintCounts;
}
my $stop_timestamp = time;
my $elapsed_time = $stop_timestamp - $start_timestamp;

my $start_timestamp = time;
$network->PrintCounts;
for (my $i = 1; $i < $steps; $i += $delta) {
   $network->Run($i, $delta)->PrintCounts;
}
my $stop_timestamp = time;
my $elapsed_time = $stop_timestamp - $start_timestamp;

my $hitness = $network->{total_hit_count}/$network->{total_msg_count};
print "# Proportion of address space filled: $fullness\n";
print "# Proportion of messages that hit:    $hitness\n";

sub numerically {$a<=>$b};

foreach my $address (sort numerically keys %{$network->{hosts}}) {
   my $host = $network->{hosts}->{$address};
   print "#hits/it", $address, "$, $host->{msg_cnt}->{rcvd}, "$\n";
}

exit 0;

sub usage {
   print "$0: Susceptible-Infected-Susceptible network worm model\n";
   print "Options:   -S seed    set PRNG seed (default set randomly)\n";
   print "          -A size    set address space size (default 65535)\n";
   print "          -V number  set number of victim hosts (default 9999)\n";
   print "          -w number  set initial number of worm hosts (default 1)\n";
   print "          -s number  set proportion of remaining victim hosts to stop at (default 0.01)\n";
   exit 0;
}
Appendix C: Descriptions of Packages Used

C.1 About NWS

“I wanted to write a system that allowed simulation of many different real-life worms. When I began, I did not understand what this entailed. At a minimum, I believed that I needed flexible configuration and setup, to allow trying different population sizes (including number and "species" of worms) and address space sizes. I also chose to have NWS actually execute "worm code" in each entity a worm infects. Actually executing code allows a worm to perform arbitrary actions just like in real life. I wanted to make any given simulation as believable as possible.

Between setup and configuration and executing code for a worm, any simulator needed to include some kind of interpreted language. I chose Perl, mainly because I had an easy time writing early versions of an SIS epidemic in Perl. I wrote the NWS framework itself in the same language that NWS users write worms.

I also chose to write NWS in a simplistic object oriented fashion. Objects in an NWS program directly correspond to real life entities that comprise a real life network: hosts, software, and messages. This should make simulations much more believable, since a user can directly count uninfected hosts or worm hosts or messages passed” (Ediger).
C.2 About SSFNet

“SSFNet is a collection of Java SSF-based components for modeling and simulation of Internet protocols and networks at and above the IP packet level of detail. Link layer and physical layer modeling can be provided in separate components.

SSFNet models are self-configuring - that is, each SSFNet class instance can autonomously configure itself by querying a configuration database, which may be locally resident or available over the Web. The network configuration files are in the DML format. They are used to configure a complete model, and instantiate a simulation with the help of the scalable DML configuration database package that is distributed with the SSF simulators.

The principal classes used to construct virtually any Internet model are organized into two derivative frameworks, SSF.OS (for modeling of the host and operating system components, esp. protocols) and SSF.Net (for modeling network connectivity, creating node and link configurations)” (www.ssfnet.org).