An Investigation into the Design and Implementation of an Internet-Scale Network Simulator

John Peter Frank Richter

December 2008

This work is submitted in fulfilment of the requirements for a Master of Science degree of Rhodes University.

Abstract

Simulation is a complex task with many research applications - chiefly as a research tool, to test and evaluate hypothetical scenarios. Though many simulations execute similar operations and utilise similar data, there are few simulation frameworks or toolkits that allow researchers to rapidly develop their concepts. Those that are available to researchers are limited in scope, or use old technology that is no longer useful to modern researchers. As a result of this, many researchers build their own simulations without a framework, wasting time and resources on a system that could already cater for the majority of their simulation's requirements.

In this work, a system is proposed for the creation of a scalable, dynamic-resolution network simulation framework that provides scalable scope for researchers, using modern technologies and languages. This framework should allow researchers to rapidly develop a broad range of semantically-rich simulations, without the necessity of super- or grid-computers or clusters. Design and implementation are discussed and alternative network simulations are compared to the proposed framework. A series of simulations, focusing on malware, is run on an implementation of this framework, and the results are compared to expectations for the outcomes of those simulations. In conclusion, a critical review of the simulator is made, considering any extensions or shortcomings that need to be addressed.

Acknowledgments

The financial assistance from Rhodes University's Dean of Research division and the Andrew Mellon Foundation towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to Rhodes University or the donor.

I would also like to thank the Telkom center of Excellence in Distributed Multimedia within the department of Computer Science at Rhodes University, and its sponsors who made this possible: Telkom, Comverse, Tellabs, Stortech, Thrip, OpenVoice, Mars Technologies, Amatole Telecommunications Services and Bright Ideas Projects.

I would like to thank my supervisor, Barry Irwin, for all of the help and support he has given me throughout this research. He has provided support and insight in numerous ways, and he has been both a friend and a teacher to me.

I am grateful to my family, whose support has enabled me to create this work. I would specifically like to express my gratitude towards my late father, Peter, who inspired me to enter this field and who continues to be a role-model in my life.

Finally and most importantly, I would like to give thanks to God, whose goodness and mercy have been a permanent blessing in my life. This work is dedicated in thankfulness to Him.

Contents

List of Figures

1 Introduction

The field of network simulation is one of a variety used in networking research. Simulators can be used for prototyping, testing and validating theories as well as recreating events for further study.

However network simulators used in research require careful development. If researchers construct a network simulator in every area in which they perform research, much of their efforts will be wasted. However, existing simulators may not always support the research that could make use of them. Furthermore, simulators which are incapable of simulations at the scale of the massive world-wide network, the Internet, are unusable for the broad body of researchers whose research focuses on the Internet on a regular basis.

The solution to this problem is to develop simulators capable of Internet-scale simulation, that use a robust set of interchangable modules for simulation that can be developed rapidly and easily. Through this, modules may be developed to create a rich library of simulated network components, available to researchers and other users of the simulator.

This research proposes the development of such a simulator, and suggests a selection of tests to determine whether the detailed simulator fulfils the expectations placed upon it. Malware has been selected as the core subject of these tests.

The simulator shall be developed, then critically evaluated using the results of the tests as a basis for determining its effectiveness.

1.1 Introduction of Concepts and Terms

Before discussion of domain specific research can be undertaken, concepts and terms that are imperative for understanding must be defined in order to avoid confusion. Concepts significant to this research are defined below - specifically, *simulation*, and the software that executes *simulations*, *simulators*, with an emphasis on two specific sub-classes of simulators: *network simulators* and *robust simulators*, and finally the subject of the simulations in this research, *malware*.

Simulation

The definition of simulation, according to the Oxford English Dictionary Online (2008), is:

> "The technique of imitating the behaviour of some situation or process (whether economic, military, mechanical, etc.) by means of a suitably analogous situation or apparatus, esp. for the purpose of study or personnel training".

Simulation is the act of preparing for or analysing a scenario by creating virtual equivalents to all the tangible and intangible objects involved in it. The simulation then follows a series of activities similar to those anticipated. After some period of virtual time has 'elapsed', the system is then reviewed and the changes noted. This allows researchers to determine likely outcomes to complex scenarios in which many variables and objects interact with the system.

Simulator

The definition of a simulator, or simulation framework, from the Oxford English Dictionary Online (2008), is:

> "An apparatus designed to simulate the behaviour of a more complicated system; esp. one for training purposes that simulates the response of a vehicle, craft, or the like, having a similar set of controls and giving the illusion to the operator of responding like the real thing."

Simulators in the context of computing are programs that can execute simulations on a computer. They accept a simulation as a series of variables and apply a given set of alteration rules to it. Once the simulation has been changed by the simulator, the variables are then reviewed. These variables can then be used to make deductions about the states of a real system once certain real transitions affect it.

Network Simulators

A network simulator is a specific implementation of a simulator (defined above), focusing primarily on computer networks. They are used for a wide variety of simulation models (including malware research, network traffic research, and new protocol testing). Because of the broad range of possible simulations, many simulators with a wide variety of features are available.

Robust Simulators

In this research, robust simulators are defined as simulators with as few restrictions as possible placed on simulations. This allows simulation modelers a large amount of freedom in adding semantic content to their models. The robustness of simulations is typically in opposition to the efficiency of the system (both in terms of memory and processor costs), as the process of allowing rich semantic content to be added to the elements of the simulation results in large overhead costs.

Implicit in the concept of a robust simulation is extensibility: robust simulators allow modelers to extend and alter the components in the simulator in order to better represent those components as they are in the real world.

The ns-2[1] and OpNet[2] network simulators are both semantically rich simulators, and Fall and Varadhan (2008) explains that ns-2 is extensible using a combination of the programming languages C and Tcl to add or change components.

Simulator Scalability

In this research, simulator scalability is defined as the ability of a simulator to simulate a large quantity of modeled objects (or a very detailed object) within limited execution time and limited memory constraints. This may include abstraction for efficiency and distribution to increase capacity.

Malware

Malware is defined by the Oxford English Dictionary Online (2008) as,

> "Programs written with the intent of being disruptive or damaging to (the user of) a computer or other electronic device; viruses, worms, spyware, etc., collectively."

Malware is a portmanteau word combining "malicious" and "software". Malware is a broad term, spanning a range of dangerous software. It typically utilises some form of communication (be it via manually transferred disks, or over a network) to transfer software to vulnerable computers. Once this is done, the effects vary depending on the purpose of the software.

Famous examples of malware include the Morris worm (the first Internet "worm", a form of rapidly propagating malware), the Blaster worm, and the I Love You virus.

The testing of the simulator designed and implemented as part of this research was primarily modeled around the spread of malware, with a focus on Internet Worm research. Much research has been generated concerning malware, specifically in analysing

[1]http://www.isi.edu/nsnam/ns/
[2]http://www.opnet.com/

the nature of various forms and implementations of malware and attempting to defend successfully against those forms.

The simulations that model malware that were run can be found in Chapter 6.

1.2 Problem Statement

Simulation is a useful research tool that can help in the development and validation of models. An important aspect that is lacking in many simulator systems is the necessity of extension and expansion in the form of *robustness* or *generality*, especially at the Internet scale. This results in the development of once-off simulations for any research which takes place that is of an Internet scale, and is outside the scope of existing components or component semantics.

This can be considered wasteful, as an entire simulator framework must then be developed to be used for a small subset of simulations, as large, well-developed simulators are ignored due to a lack of capability.

1.3 Proposal: A Robust Network Simulator

This research proposes a solution to the challenge of large-scale robust and general simulator development. The development of a simulator solution that can make use of these traits offers value in its ability to provide this rapid simulation development, while allowing for a rich existing component set and an easily extensible framework for further component development. This addresses the problem stated in Section 1.2.

In introducing a complex system such as this, it is essential to initially state and continually review the work with two major considerations: the value of the system to the user, and the capabilities expected of such a system.

1.3.1 Value

A robust network simulator system has many uses, depending on the nature of the user. Particularly in the case of robust simulators, where the simulation domain is large, the system can be used by a wide variety of computer users for a broad range of tasks. Users that can expect to find the maximum use from such a system are system or network administrators, software developers that will interact with networking at some level, and researchers.

For System Administrators

System or network administrators can use robust network simulators in designing their networks, testing when they encounter problems, and in preparing for or countering malicious activity that may influence their networks. Examples of research that makes use of simulations for the purpose of protecting or optimising networks include Zou and Gong (2004) and Baccelli and Hong (2003).

Through the use of a simulator, they may determine the effects of change, model already-existing changes to see what effect they have, and prepare for scenarios on their networks that they anticipate occurring.

For Network Software Developers

Software developers that author programs that interact with a network can make use of a simulator to fully determine the effect their programs will have on a network. By simulating a network with their software running, they may be able to find deadlocks, traffic problems, possible exploits and test problems in their code when networks are not structured or operating in the way in which they initially anticipated.

In Castaneda et al. (2004) the authors devise a system for countering Internet worms, and use simulation to determine its effectiveness.

For Researchers

Researchers use many simulators already, such as ns-2 and SSFNet (detailed in Section 2.6) in order to test hypotheses and generate data for conjectures. Robust simulators allow a single simulation system to be used for a wide variety of simulations that a researcher might require. By reducing the number of simulator frameworks, researchers can use a single set of skills, knowledge and code for their work, speeding development and simplifying their work.

1.3.2 Capabilities and Properties

A complex system such as a simulator can be created in a variety of ways. When several conflicting options in development are being considered, it is important that clear priorities are established so that the resulting system performs optimally. With this in mind, the following capabilities and properties are necessary in the system:

Scalable

The simulator must be scalable. Without scalability as a core property of the system, simulations will be strongly restricted to a finite quantity of semantic meaning, resolution, or addressable simulated objects in the models, or the basic system requirements will have to be incredibly large. While keeping scalability in mind, it is also important to consider the trade-off of optimisation to accuracy. Accuracy is typically lost when efficiency is improved in systems, but in the case of simulators that require highly accurate results, this should be avoided at all costs.

Scalability can be mitigated by distributing the execution of a simulation, however the goal of this work is for robust network simulators to operate on single hosts.

Arbitrary Resolution

Because of the varying needs of users (shown in Section 1.3.1), the resolution of a network simulator needs to be variable. As part of the concept of a robust network simulator, it is imperative that broad concepts (such as a 'worm', or even 'the Internet') can be modeled as an individual entity for users with high-level simulation concepts. It is also necessary that very low-level concepts (such as individual packets, or even voltages in cables) be able to be modeled, for the use of researchers and others who need to explore networking at that level.

Though it is not necessary, it should also be possible within simulations to be able to change resolutions. For example, a conceptual 'network' object in a simulation should be able to be broken into a variety of 'host' objects as necessary. This is a complex task, and should not be a required aspect of a simulation, but rather an optional feature should the modeler require it.

Semantically Scalable

Modeled objects within the simulation should be able to contain an arbitrary number of properties and associated values. It should also be possible for objects of a similar type to contain different amounts of semantic information - some objects in the system of a certain type should be able to be simply defined, while others should be complex and have a great deal of associated information.

Distributable

Though it must be able to run simply on a single system, the option of distributing the execution and memory load must be available. Because of the necessity of scale, mentioned above, very large networks with large quantities of semantic content are possible objects of simulations. The memory and processor requirements for such networks could easily become large enough that single-computer simulation is no longer feasible. In such a case, the simulator should be designed with distributed or grid computing in mind, to take advantage of large computing farms or clusters.

1.4 Goals

In order to guide this research, a number of goals need to be stated and revisited as development takes place. These will serve to keep the research within useful boundaries, as well as giving an absolute set of outcomes to be achieved by the end of the research. These goals are listed here, with justification:

1.4.1 Recommendations

A set of recommendations for development of robust simulators (similar in design to the simulator described in Section 1.3) should be detailed. These should cover the requirements of such a simulator, as well as displaying research that might aid in development, and expanding on that with practical development guidelines.

1.4.2 Simulator

As a proof-of-concept, a simulator should be developed using the recommendations from Section 7.2.1. This simulator must fulfill the criteria stipulated in the recommendations, and use the advice the recommendations give. It should be documented, starting at the development stages and continuing through to its practical uses.

1.5 Research Methodology

In order to properly develop a solution to the problem statement in Section 1.2 and thus meet the goals in Section 1.4, it is imperative that a proper research methodology is followed.

A review of research (both current and dated) should take place and will be mentioned in Section 1.5.1. Experiments on the proposed simulator should be run to determine the effectiveness of the simulator for modeling networking and malware, and is mentioned in Section 1.5.2.

1.5.1 Literature Review

A review of the literature on the subjects that influence this research must take place. Specific areas that will require research include simulation (directly, with research on simulation itself, and indirectly, with research that makes use of simulation as a research technique), networking and malware.

This literature review can be seen in Chapter 2.

1.5.2 Experiments

Simulations should be developed (conceptually initially, then programmatically), then executed upon the simulator developed as part of Section 7.2.2. The simulations should thoroughly test the simulator, particularly in terms of robustness (its ability to represent a broad set of concepts), efficiency (its ability to complete simulations in reasonable periods of time), scalability (its ability to represent very broad concepts accurately while maintaining efficiency) and accuracy (the problem domain simulated should approximately match observed data).

The experiments used in this research are detailed in Chapters 5 and 6.

1.6 Document Overview

In this chapter, we have reviewed the problems inherent to malware simulation, and a simulator designed for rapid simulation modeling has been proposed.

In Chapter 2, available textual resources are discussed, focusing on texts that span the areas of malware, simulation, and networking.

In Chapter 3, design considerations and implementation details for the proposed simulator are covered.

In Chapter 4, the design of simulations (on a more generic level) are discussed. Prototype simulations that were tested are used to show challenges that were experienced in the development process, and Internet worms are introduced as the core subject of simulation.

Chapter 5 introduces simulations, simulating network fundamentals.

Chapter 6 continues simulation, covering Internet worms and more advanced malware simulations.

Finally, the document is drawn to a close in Chapter 7.

2 Literature Review

2.1 Introduction

Constructing a simulation framework capable of large simulations and that can serve a variety of purposes in research is a non-trivial problem - Paxson and Floyd (1997) entitled their paper "*Why We Don't Know How to Simulate the Internet*", and detail these challenges.

As the core concept of this work deals with the production of a computer network simulator, it is imperative that a deep understanding of the workings of the Internet be available. Section 2.2 considers some literature on computer networking (at a broad general level) as it pertains to this research, focusing on networking and the Internet as subjects of simulation.

Section 2.3 continues to consider literature regarding networking simulation, focusing on present works that use simulation as a research tool. Where simulation has been used in research, its value is considered and noted for use in development.

In Section 2.4, Internet worms and other forms of network-propagating malware are discussed. Network security is a common goal of research, and could be used as a focus for a network simulator.

Section 2.5 considers research on Internet Worm simulation, focusing on simulation used for understanding the malware and Internet events considered in the previous section.

Before development can begin, the paradigms, benefits and drawbacks present in current simulation frameworks need to be extracted and considered. Where aspects of these simulators allow for broad, efficient simulation, they should be examined and recorded for later use. This evaluation of existing network simulators takes place in Section 2.6, concluding the chapter.

2.2 Networking and the Internet

Before any development of network simulators may take place, an understanding of the fundamentals of computer networking is required. In this section, the fundamental notions of modern networking and the Internet are considered. Thereafter, some more recent research into the way modern networks operate is mentioned and discussed.

2.2.1 History of the Networking and the Internet

According to Leiner et al. (1999), the Internet was first mentioned by J. C. R. Licklider of MIT in Licklider and Clark (1962), and was developed at DARPA and commissioned by the United States of America's Department of Defense (DoD). Initially termed the "ARPANET", it grew rapidly, with an increasingly large body of researchers working to improve the communication protocols. In 1972, ARPANET was publically presented at the International Computer Communication Conference (ICCC), and electronic mail (E-mail) was introduced.

Between 1973 and 1976, Vint Cerf was contracted by DARPA to develop and mature the TCP/IP protocols at Stanford. Significantly, according to Zakon (2006), the Department of Defense declared the TCP/IP protocol suite to be their standard protocols. This resulted in widespread adoption of the protocols, laying the foundations of the modern Internet.

The Internet grew out of ARPANET, and the modern protocols that are still used today were introduced - the Internet Protocol (IP), detailed in Information Sciences Institute (1980a), and the Transmission Control Protocol (TCP), detailed in Information Sciences Institute (1980b). These take advantage of the layered nature of networking protocols, defined in International Telecommunication Union (1994).

The "Open System Interconnect" (OSI) stack is introduced in International Telecommunication Union (1994), and introduces the idea of layered networking protocols to allow for modularity. By keeping the various component parts of networking separated into layers, it is possible to transparently "wrap" high level communication in lower level protocols which hide the specific communication details. From the opposite perspective, lower level protocols "wrap" higher level protocols as data, ignoring the meaning of the information they contain, acting entirely as the communication medium.

The OSI stack can be visualised as shown in Figure 2.1, presenting the communication protocols of the Internet as a series of layers. The higher layers are "wrapped" in the lower layers which describe the form in which communication takes place.

2.2.2 Networking Research

Due to the large quantities of information that are communicated on the Internet, contention for bandwidth is a common problem: Jacobson (1995) introduces their

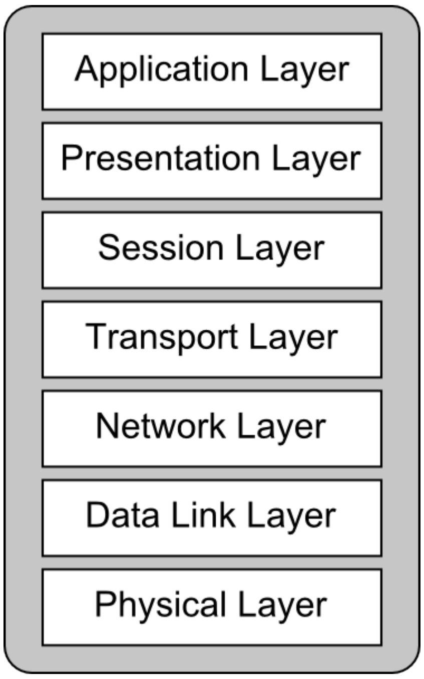

Figure 2.1: OSI Stack

research by stating: "Computer networks have experienced an explosive growth over the past few years and with that growth have come severe congestion problems". Because of this, much research has taken place to determine the best or fairest way of distributing available communication resources: Baccelli and Hong (2003), Dutta et al. (2002) and Feldmann et al. (1999) are some examples of this.

A common theme in this research is the use of simulation to investigate proposed solutions to communication difficulties. Thus, a common theme in computer network simulation research is modeling bandwidth, as bandwidth and congestion research will require this property of networking to be simulated. Discussed below are papers that discuss networking without the use of simulation, but that include representational models to emphasise the validity of their research.

Savage et al. (1999) point to the inefficiencies of modern IP. Discussing the initial assumptions made about Internet routing, they show that in many cases more efficient routes exist between ASs (or Autonomous Systems, a term for a group of networking resources controlled by a single entity) that could be exploited, but that are not chosen due to the distance between nodes.

They name four key inefficiencies in the present means of routing:

- Poor routing metrics
- Restrictive routing policies
- Manual load balancing

- Single path routing

They propose an alternative model for routing, which they name Detour, which intelligently chooses routes between hosts, suggesting that it would result in significant improvements in performance over the Internet.

The Detour model could be modeled using the proposed simulation, either as a representational prototype (See Section 2.3.2), or to test its efficiency. Any resulting trade-offs and side-effects, if any, could be observed prior to its deployment, resulting in a more thoroughly tested model and a greater degree of certainty of its success.

Baccelli and Hong (2003) use a “flow” abstraction of traffic to better model bandwidth. They go on to use physics equations from the field of fluid dynamics to model the interactions between TCP and UDP flows, and use the network simulator ns-2 (discussed later in this chapter) to validate their models.

Zhou and Mondragon (2003) deal primarily in statistical modeling of the Internet. The authors discuss the ‘Rich Club’ phenomenon of the Internet - how nodes in a network that are already well connected will gain the majority of new connections, while less popular nodes gain fewer and fewer. After doing this, they discuss other popular models of the Internet (Inet-3.0 (Winick and Jamin (2002)), Barabasi-Albert (Barabasi and Albert (1999)), and the Generalized Linear Preference (Bu and Towsley (2002)) models), and compare it to their own alternative, the Interactive Growth model.

In closing, they state that they "expect the model to be used in simulation-based research for the Internet traffic engineering." In their own research, however, simulation could be used as a means of both testing and validating their proposed model.

Comparing the statistical modeling of Zhou and Mondragon (2003) to the representational form of simulation modeling allows a researcher to understand the differences and important roles that each type fills in research - and these two forms of network simulation are examined in the following section.

2.3 Network and Internet Simulation

There is a broad body of literature in which network simulations have been used to test hypotheses, such as those mentioned in Section 2.2. In this section, papers that are relevant to the research presented in this thesis are detailed and commented upon.

In Sharif et al. (2005), it is stated that simulations that take into account packet-level models of networking and worm propagation more accurately model the complexities of worms than those that use analytical means to do so. While packet-level models are more detailed, they require significantly larger computational overhead than those using analytical models. While the paper will be discussed in more detail in Section 2.4, the differences between analytical (and mathematical) models for simulation, and packet-based (or representational) models are underlined in this paper. With this in mind, the remainder of this section is split into Section 2.3.1 on papers that consider analytical simulation models and Section 2.3.2 on papers that consider representational simulation models.

2.3.1 Mathematical Models

Salamatian and Vaton (2001) focus on mathematically modeling the state of a network using Hidden Markov Models (HMMs). HMMs are equations from the field of statistics, which make inferences about unknown states in systems by analysing the outputs of those systems. This paper uses packet delay and loss to make inferences about the states of communication channels (in this case, very simplified states - congested and non-congested - are used).

This paper presents interesting models of the Internet, and shows a means of projecting hard-to-acquire knowledge from trivially acquired data. Using information and models like those used in this paper, it is possible to acquire data about incoming communication and derive models about the most likely states of massive, hard to know systems (like the Internet).

In "Mathematical modeling of the Internet", Kelly (2001) discusses the rate of packet flow in networks. The author outlines mathematical models and quality of service systems for the Internet, specifically focusing on TCP.

The mathematical models shown in this paper could be representationally simulated by altering the TCP portion of a simulation system and observing the changes in the overall network that is simulated. By doing this, the effectiveness and efficiency of the new system can be measured. Once the benefit of this new technology can be measured, the real value can be used for more accurate decision-making.

Baccelli and Hong (2003), Zou et al. (2002) and many other researchers begin their research by proposing a mathematical or statistical model for the simulation of computer networks. They then go on to validate their models using a representative network simulator such as ns-2 or SSFNet.

2.3.2 Representational Models

In Baccelli and Hong (2003), the introduction states that, "It is well known that the packet level simulation of TCP over IP with tools like ns-2 or Opnet is currently not possible for large populations of flows and/or large numbers of links/routers". This points to the necessity of dynamic abstraction in the development of a simulator - the necessity of a simulator to be able to represent various degrees of abstraction, and if necessary switch between simulating at those different levels.

This paper dealt specifically with routing, defining traffic in terms of flows - an abstract concept used to describe TCP connections and other form of transactional communication.

In Bai et al. (1999), the authors use a popular simulator (SSFNet, detailed further in Section 2.6) to simulate the effect of two protocols on a wireless network. They proceed to optimise the TCP sliding window for wireless networks by observing the errors that occur due to the conflicts caused by the two protocols (TCP and ARQ, a radio link-layer protocol). This demonstrates the necessity of components that support the simulation of wireless networking in a simulator, expanded upon in the proceeding work.

Chen et al. (2002a) describe a novel idea for distributing routers in an ad-hoc wireless network to minimise power use. Their design operates by ensuring that every contributing ad-hoc host is capable of routing to the Internet, while still using the minimum number of routing nodes, periodically changing routing nodes to ensure fair distribution of routing load (with a priority on overall connectivity). They use the ns-2 simulator to demonstrate the effectiveness of the system. This demonstrates a further feature necessary in network simulation - the ability to add (without significant alteration to the simulator) additional 'layers', requirements or attributes to parts of the system - in this case, 'physical distance' is an important part of the simulation.

Dyer and Boppana (2001) confirm the necessity of being able to represent 'physical distance' in a simulator that is robust: the authors use the ns-2 simulator to simulate the performance of TCP, evaluating a variety of routing protocols available for wireless or mobile computing. They performed the simulation using a 1000x1000 grid for location of nodes - indicating that modern research requiring robust network simulators include wireless networking as a part of their simulator's capabilities. This could be modeled in a simpler manner with a simulator that is capable of containing location attributes to their modeled hosts - distance can be calculated easily once physical coordinates of the nodes are available.

In Cowie et al. (1999), the future plans and open problems that they describe are:

- Visualisation challenges
- Tools for generation and validation of network topologies
- Scalable data collection facilities
- Multiresolution tools for sensitivity analysis

All of these are significant problems to be considered when developing simulators. Visualisation can occur either as during- or post-processing, and gives the user an easy and simple way of determining the state of the simulation. Tools for generating networks allow for rapid prototyping and simulation development. Data collection facilities allow the system to determine how patterns (such as traffic, distribution of node type, etc.) should be modeled using real data obtained from a network. Multiresolution simulations are discussed more thoroughly in Huang et al. (1998), and provide a means of simulating broadly (and efficiently) using abstractions while still maintaining a high degree of accuracy.

Dutta et al. (2002) discuss means of rapidly designing networks using scenario prefiltering - a technique that allows for efficient simulations, even though they are complex. This is a technique that could be adapted for use by simulators that intend to do massive- (e.g. Internet-) scale simulation.

By excluding parts of the network dependant on the amount of bandwidth they are simulated as transferring, it is possible to limit simulation to only ‘interesting’ parts of the network, making the simulation process more efficient. This could, however, result in a loss of accuracy.

Dutta et al. (2002) state:

> “Packet level simulators, such as ns-2, simulate the network as a series of discrete events, requiring a number of events proportional to the number of packets generated by the network. Although simple simulations can be run quite quickly, simulating scenarios with many nodes and at high traffic rates can easily become quite time consuming. Understanding the behavior of the network may require many scenarios with alternate traffic or configuration choices. Often many of these scenarios are not interesting, either they are very overloaded (and so not a sensible operating point), or they are very underloaded (and so not providing insight into the network’s performance).”

This underlines the importance of extracting the significant events in a simulation, and ignoring those that are not important.

Farber et al. (1998) analyse the dial-up traffic patterns at the University of Stuttgart, using data gathered, specifically focusing on dial-up holding times, login times, traffic load and interarrival times. While the technology used for this paper is dated, the concepts (statistical modeling, using gathered data for simulation modeling) remain relevant.

The authors effectively use data gathered to generate a statistical model and evaluate performance. This is significant, because statistical models must play an important role in simulation development. If an analytical model is presented for testing, a simulator must be capable of using its available components to best representationally model the functions presented.

In Feldmann et al. (1999), models of user and network traffic are tested for accuracy using the ns-2 simulator. The authors use low-level protocol variables in their simulation. Simulators should be capable of altering variables at a low level (such as changing parts of a low-level protocol) easily, and see the effects of the change at higher level (such as seeing high-level protocols which rely on the affected low-level protocols changed). This can be seen again in Joo et al. (1999), where common assumptions of traffic patterns used in simulation are challenged. By using two different cases of underlying bandwidth patterns, they show the differing effects that could be observed in higher-level protocols.

As stated in Section 1.3.1, an important use of simulators is in prototyping and pre-release testing. In Garetto et al. (2001), a new form of TCP performance measurement is discussed, and ns-2 is used for simulation to test the quality of the measurement system. They base their research on previous work which successfully demonstrated the positive effects of closed queueing networking for measuring TCP connection performance, but which had failed to perform any simulation of this change in measurement.

This indicates the necessity of simulators as tools for prototyping new concepts, as well as tools for validating existing research which uses analytical models instead of representational simulation.

In Hanle and Hofmann (1998), three alternative protocols for multicast traffic on the MBONE network are compared, and simulation comparisons are made on ns-2. The paper comments on the robust set of existing components (particularly protocol forms) in ns-2, and proceeds to use these components with an added testing layer of these multicast protocols. Simulators should have prebuilt component sets, while still allowing users to develop their own components for specific testing purposes.

Henderson et al. (1998) test and simulate various hypotheses about TCP congestion, notably imposition of a constant bit-rate policy on a network (and its effects on a long return-trip time connections). This shows the necessity of packets in a simulation - while it is obvious that nodes and the connections between them need to be detailed, this details the necessity of containing TTL (i.e. distance) and time information in order to adjust the behaviours of nodes for efficiency.

In Hofmann et al. (1999), improved means of caching streaming media on the Internet are proposed. Several techniques grouped under an architecture named 'SOCCER' (Self-Organizing Cooperative Caching Architecture) are suggested. They use ns-2 to prototype this architecture. The suggested techniques and components in the framework are:

- Stream Segmentation (sending part of the stream, not all of it at once)
- Dynamic Caching (if a cache request for an already streaming object is made,

send with the current connection and send the differential ‘patch’ afterwards)

- Self Organizing Cooperative Caching (by intelligently and dynamically coupling caches, they can send objects to various receivers efficiently)

These components are all modifications of the node component, and can be grouped under the ’sending’ and ’receiving’ parts of that component. This outlines the necessity of those components being modular or at least easily editable if the context of the simulation requires it.

Modeling the topology of the Internet is a fundamental and non-trivial task needed in an Internet simulator. In Jeremie (2004), methods for modeling the topology of the Internet are discussed.

Initially, the basics of TCP/IP communications are discussed, with a focus on routing, inter-host communication and the present organisation of the Internet. This is followed by a discussion of several existing projects, covering network topology discovery projects and statistical model generators. It then goes on to cover currently accepted properties of the Internet.

The core of the paper regards the use of CAIDA’s Skitter data (Cooperative Association for Internet Data Analysis (2008a)), focusing initially on the visualisation of the data, then moving on to the statistical analysis and discussion of the definition and modeling of Internet routes.

In this paper, Paxson and Floyd (1997) discuss the difficulties of massive-scale network simulation. The chief difficulties that they describe are:

- The heterogeneity of the Internet, specifically:
  - The different connection types
  - The different types of congestion found
  - The different network topologies and link properties
  - The different protocols used
  - The different applications which are run on the various nodes
- The constant change in the nature of the Internet, effectively making it one ’large, moving target’
- The massive scale of the Internet, where events which are rare at the local level occur often at some point in the broader network

They then go on to suggest that the best way to cope with these problem is to focus on the system’s invariants: certain facets of the Internet, such as statistical distributions of connection regularity, packet arrival, and the log-normal distribution of connection size and time. They also point out that it is possible to derive some parameters of the Internet: however, no single simulation can provide them. As Paxson and Floyd (1997) say: "If you run a single simulation, and produce a single set of numbers (e.g., throughput, delay, loss), and think that that single set of numbers shows that your algorithm is a good one, then you haven't a clue."

Finally, they discuss the Vint project, an attempt at a successful simulation.

Network simulation suffers from scaling and granularity concerns: if a network is too highly detailed, performance impracticalities arise. However, high-level networks fail to include details that granular simulations can perform. A solution to this problem is proposed in Rao and Wilsey (2001): multi-resolution network simulations.

Multi-resolution network simulations allow for levels of abstraction and detail to be introduced into a simulation such that no uniformity in granularity level is required. This powerful tool allows networks to be simulated with varying amounts of detail, yet still allows practical communication between these components.

Combining this with Dynamic Component Substitution (DCS), simulations can be designed that change resolution at run-time or at compile-time - simulations that allow dynamic abstraction to compensate for load, to allow minute details to be modeled in certain sections of the simulation and have these affect the broad generalisations that are running in other areas of the simulation.

2.4 Internet Worms and Malware

In "Aggressive Network Self-Defense" by Wyler et al. (2005), several authors relate controversial short fictional accounts that argue and explain the ethics behind the concept of "strikeback" - using aggressive or passive tactics to compromise the computers of attackers, forcing them to cease any malicious activity.

One of the forms strikeback takes is a "helpful worm" - worms which make use of backdoors left by other worms, to infect the system, delete the malicious worm, then delete themselves after propogating and closing the backdoor. Helpful worms are a form of malware countermeasure, and so are also detailed in the next section.

The idea that helpful worms can be used to counter malicious worms could be used as a good test of a simulation system. Both worms would (initially) create a massive amount of traffic, and the relative success of each would be an interesting means of measuring the effectiveness of the concept.

A theoretical construct that is similar to these worms is found in the paper "A Predator-Prey Approach to the Network Structure of Cyberspace," by Gorman et al. (2004).

Zou et al. (2003) proposes a Dynamic Quarantine System that may help to slow worm propagation, based on the principle of 'guilty until proven innocent'.

The system uses an unnamed anomaly detection system that guesses when network activity from a host could be construed as being caused by a worm. Even in the case of possible false positives, the system 'quarantines' the packets that have been sent. They are not received by the hosts that the system is defending - instead the packets are held temporarily.

At this stage, the authors comment that in a truly secure system, a human supervisor should inspect the activity in order to determine whether it is truly a worm, or merely a false positive. The noted problem with this is that many hosts may be behaving in a 'worm-like' manner, and the length of time it would take for a person to inspect the quarantine may result in loss of connectivity to critical or harmless systems.

Because of this, the final system introduced in this paper proposes simply holding quarantined packets for a period of time, before finally allowing them into the system. Because of this, worm propagation is significantly slowed, allowing human countermeasures to be developed in a reasonable time.

This system would be particularly useful in countering so-called 'Warhol' or 'Flash' worms (discussed in Nazario (2003)) that spread at rates faster than any other kind of worm found so far.

2.4.1 Worm Countermeasures

Zhang et al. (2004) propose a system of quarantines to counter worm propagation. While they do not dwell on the details of the quarantine, they use the standard 'Susceptible-Infected-Removed' Markov model to show the advantage of a quarantine system in reducing the speed at which a worm can propagate. This shows an interesting means of host classification when worms are simulated - hosts can easily be classified in one of these three categories, and host behaviour can defined in terms of which state it is currently in.

In Coull and Szymanski (2007), it is recommended that a worm countermeasure be installed over a large portion of the Internet. It continues to describe how the countermeasure system would use a reputation system for analysis of worm attackers. The apparent challenge in the implementation of this is the installation of a malware countermeasure package on such a massively heterogeneous Internet, controlled by a diverse range of interests. This would appear to make this recommendation untenable. Furthermore, its effect would only be significant if these portions were towards the core of the Internet, were speed and bandwidth are priorities - scanning for worm packets, however efficiently done, would cause a noticeable effect on the Internet.

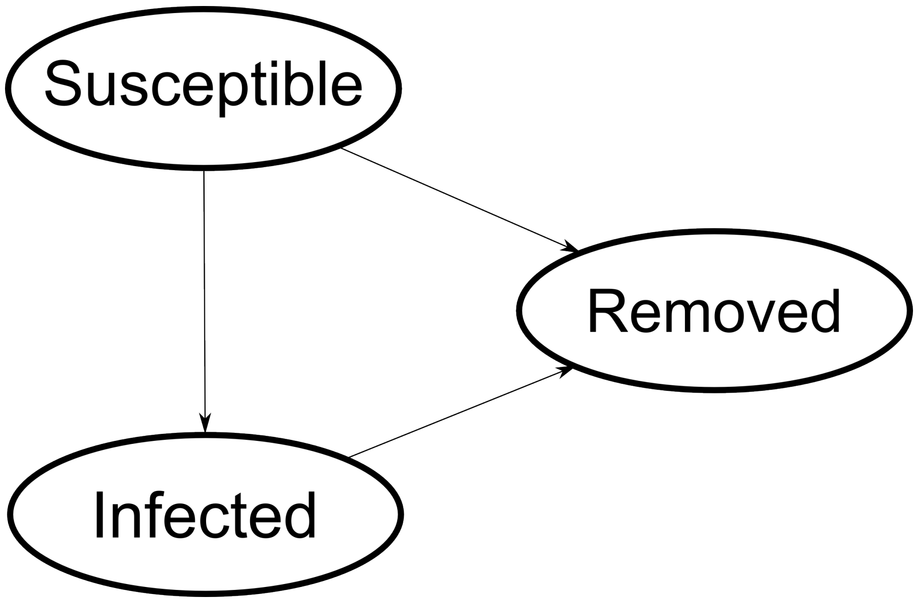

Figure 2.2: Susceptible-Infected-Removed model

In terms of simulation, this paper suggests a home-grown simulation system. A generic, robust simulator could not only perform similar simulations, but could easily be extended for further testing in the domain. Reputation scores are a simple addition to most simulations, and can be used to represent trusted or untrusted networks.

The use of anti-worms to counteract malevolent worms is a controversial example of "Strike Back" (also mentioned in Wyler et al. (2005)). In Castaneda et al. (2004), the authors state the major concerns, and cite some famous examples of anti-worms that have been shown to have detrimental side-effects (notably Welchia). They then go on to simulate the effects of an anti-worm. The simulations they run, however, are analytical as opposed to representational - they use a series of mathematical models. Their results show that anti-worms are not, at present, an appropriate solution to malevolent worms, but are still an intriguing concept.

Using SSFNet, Briesemeister and Porras (2005) propose and simulate a method for worm detection and countering. They use a broad catching strategy ('group defense strategy') to ascertain the origin and signature of the worm, then rate-limit to slow the spread. This could easily be simulated in a robust simulator engine (and has been, see Section 4.2.4), particularly well if connection objects have been customised to more accurately simulate rate limiting.

2.5 Internet Worms and Malware Simulation

Internet worms are an ongoing threat to the stability of the Internet - Castaneda et al. (2004) describe Internet worms as "terrorizing the Internet for the last several years".

As a result, research is taking place to better understand worms, to prepare for them, and to consider new forms of worms that might threaten the Internet. One of the tools used in this research is simulation - it allows security developers and researchers to prototype concepts safely, more accurately predict outcomes when additional effects are added to a known system, and experiment with dangerous concepts without any risk.

One project (in development) that is attempting to make use of massive-scale simulation for network security research is ISEAGE, developed by Iowa State University Information Assurance Center (2008).

In this section, Internet Worm research that makes use of simulation is discussed, along with research specifically aimed at challenges in Internet worm simulation development.

Before any specific “real” Internet worms are studied, research that focuses on worms conceptually is shown in Section 2.5.1.

This is followed by an overview of research on the Code Red vII worm, which has received a lot of attention from researchers, in Section 2.5.2. Section 2.5.3 continues by summarising the research on worms other than Code Red vII, and Section 2.5.4 considers the research about worms in general, without any specific focus on any individual worm.

Finally, Section 2.5.5 considers worm simulators that are currently in use.

2.5.1 Mathematical Models of Worm Behaviour

Chen et al. (2002b) introduce a mathematical model of worm behaviour: The AAWP or Analytical Active Worm Propagation model.

In this paper, Chen et al. (2002b) present a model of worms that randomly scan through IP space, actively searching for hosts to infect. Factoring in ‘hitlists’, or initial lists that worms are seeded with for hosts that the author particularly wants infected (whether they have high bandwidth, are known as exploitable, or for some other reason), this paper presents the probability calculations of host infection by neighboring nodes.

The authors spend some time comparing the AAWP model to the statistical model that was first used for modeling worm spread, the Epidemiological model (). One of the differences that is notable is the importance of discrete time in infections - a worm does not start spreading until the host is entirely infected - a factor not taken into account in the Epidemiological model. They then go on to parameterise their model for simulating the Code Red vII worm.

This model is well suited to simulation - and as commented upon in Section 2.3.1, it is important that simulators be capable of easily adapting a mathematical or statistical model into a representational one.

2.5.2 Code Red vII Worm

Wagner et al. (2003) summarise the authors' experiences simulating the Code Red Internet Worm. They also open with an excellent argument for the creation of Internet Worm simulations:

The benefits of predicting worm behaviour are numerous:

- Better understanding of the behaviour of worms observed in the past
- Estimations of a worm's threat potential
- Estimations of the impact of future worms on the Internet
- Basis of the design of detection mechanisms for worm spreading
- Determination of parameters relevant for worm characterisation

They then continue to explain different methods for studying worms: mathematical models, sandbox testing, study of observed data, and simulation.

Following this, they go on to detail the parameters they used for a simulation they developed. They used two peer-to-peer networks as a measure of distribution of bandwidth on the internet - as an example, because 32% of Napster's users connected at 64 kb/s, they assumed that 32% of the Internet did the same.

They then go on to run various simulations, focusing mainly on bandwidth and latency measuring, and show the similarities between these simulations and the observed results.

Zou et al. (2002) focuse on the Code Red Worm (specifically, Version 2) that was inflicted upon the Internet in July 2001.

The authors propose that in order to statistically model the spread of the worm, two factors need to be taken into account: from the pool of susceptible computers (S), with some infected by the worm (I), computers can be rebooted, effectively moving them back to the susceptible pool. Infected computers can also be patched or the worm can be neutralised, removing them from the infected pool or susceptible pool into the removed pool R.

Through the use of a simple McKendrick model (originally defined in Kermack and McKendrick (1927)), the authors define a series of equations that model the growth of the Code Red Worm, taking into account the slowing of growth as the population of pool S becomes saturated and moved into pool I. By using negative growth parameters for susceptible hosts, positive and negative growth parameters dependent (respectively) on the number of susceptible hosts and the number of removed hosts for the infected pool I, and a positive growth parameter for the pool of removed hosts, independent of the number of hosts in either pool (as a host can be patched without being infected).

Possible extensions to investigate in this project are the separation of growth parameters for the removed pool - people may be more likely to patch if their hosts are infected.

When the Code Red vII worm was first discovered on the Internet, one noted side-effect was an increase of BGP routing traffic on the Internet. In Liljenstam et al. (2002), the possible reasons for this are explored.

The simulator they use is at a high level, referring to ASs or Autonomous Systems. As ASs can scale in size from single computers to large networks, this simulation results in large quantities of abstraction.

In order to maintain efficiency, this simulation simulates a small number of ASs in detail, and considers the remainder of the Internet to be a single, large AS.

By using the statistical General Epidemic Model, the propagation of the worm is modeled. Of interest is the simple idea that the rate of spread of infection is proportional to the product of the two population sizes: the susceptible and the infected.

The vulnerability in the IIS webserver that the Code Red vII worm exploited can be found in Microsoft (2003c), and the worm itself has been analysed by Friedl (2001).

2.5.3 Other Worms

Bailey et al. (2005) dissected the Blaster worm that was found on the Internet in August 2003. Initially focusing on a broader overview, they continue by discussing the workings of the worm, and variants of the worm that it uses. Finally, they conclude by observing the Blaster worm after the initial spread, noting especially how it is still "alive" and active on the Internet.

The Blaster worm was also studied in Castaneda et al. (2004), and the official analyses of its propogation can be found in Dougherty et al. (2003), Knowles and Perriott (2003) and of the DCOM RPC exploit that is uses in Microsoft (2003a) and Microsoft (2003b).

The Welchia worm is also studied in Castaneda et al. (2004), and its propogation detailed in Perriot (2008).

The Witty worm is studied in Stewart (2004) and Shannon and Moore (2007), and the vulnerability used is detailed in Gatti et al. (2004).

2.5.4 General Worm Literature

Weaver et al. (2004) focus on the degree of parameter change used in simulations of worms: most simulations scale down the size of the address space of the Internet in order to allow for more efficient simulation. The authors argue that while this remains relevant, in order to draw useful data from simulations, other parameters need to be scaled as well. Chiefly, their argument focuses on the scan-rate of the worm dropping off faster, and the problem of reducing randomness to fit the new scale.

In Zou and Gong (2004), a model for the spread of email worms is suggested. The authors then go on to mention the simulations that they have run using this model.

Though this paper focuses more ontheory and doesn't dwell on the simulations involved, the important element is the OSI stack layer of the level of simulation: by creating simulations at the top of the OSI stack, they have abstracted a significant amount of underlying detail.

This highlights a need for robust simulators: high-level protocols must be able to be simulated in a similar way - should an application-level protocol need simulation, the framework should be robust enough to support it.

On a more generic level, Gorman et al. (2004) discuss the modeling of malware on the Internet using methods traditionally associated with biological sciences.

In traditional predator-prey models, it is shown how an equilibrium between two groups of creatures, a predator species and a prey species, is established. When few prey animals exist, predators starve and die. When few predators exist, prey animals grow rapidly. Due to the increasing amount of prey, the numbers of predators swell. Due to the increasing number of predators, prey population shrinks. This returns us to the initial state of the example, and shows the established equilibrium.

Gorman et al. (2004) use the metaphors of predator and prey in an online context: Predators are hackers and malicious software. Prey are internet-enabled hosts that can be compromised by the predators.

Though significant differences exist between biological predator-prey models and electronic predator-prey models (such as the speed of growth of the populations, or the need for certain types of predator to consume specific types of prey), the model is still considered viable for simulation.

They then go on to outline a methodology for simulating the predator-prey relationship, with a small percentage of available hosts acting as prey, and predators introduced into the simulated network system.

2.5.5 Worm Simulators

In Liljenstam et al. (2003), the authors develop and document their worm simulator, the "DIB:S/TRAFEN" system, discussing its possible effect on the Internet and the theory behind its function.

This two-component worm-countermeasure system works by breaking the concept of a worm into two distinct properties:

1. The random-scanning nature of (some) worms
2. Packets from a single worm carry some signature specific to that worm

Through these two properties, the system works as follows: The DIB:S (Dartmouth ICMP BCC System) system watches certain unassigned IPv4 Internet addresses. Because the non-existent hosts at those addresses can have no practical functions, any attempt to access the address can be concluded to be either useless or malicious. Because of this, any packet arriving at that location is immediately tagged to be watched, and (because worms can generate massive amounts of traffic), the address is ignored for a small period of time.

If these watched addresses are distributed evenly across a large amount of IPv4 address space, then the packets received can be considered to be a good statistical sampling of random packets sent across the Internet.

The TRAFEN (TRacking And Fusion ENgine) then observes these alerts, and attempts to find common signatures from them. It does this through a series of constantly updated hypotheses.

Through this combination of systems, large-scale random-scanning events on the Internet can be rapidly detected and countermeasures can be developed. This could be simulated using a robust simulator, and would take a form very similar to that used in network telescopes, considered in Section 2.4.1.

This paper also takes into account the development of a simulator that can be used to test the DIB:S/TRAFEN system, then discusses experiments done with the simulator.

Finally, the authors compare their simulated data to observed data and conclude that their system would be effective as a worm deterrent.

2.6 Network Simulators

As discussed in Section 2.3 and Section 2.5, simulation is an important part of network research, and it has been a vital part of Internet worm research for many years. As a result, several notable simulators have been developed. These simulators have a broad application domain - they typically attempt to model (in detail) many forms of network traffic, and allow the user to change parameters according to the needs of their simulation.

Several simulators are popular amongst researchers, and they are discussed here:

2.6.1 ns-2

A popular network simulator, ns-2[1] is a free, open source network simulator that uses the C and Tcl programming languages to develop simulations. As part of the ns-2 suite, a variety of tools are made available to simulation developers, such as the simulation engine (for which the suite is named) and nam, the simulation visualiser.

The following research uses ns-2 as a core component, and displays the simulators capabilities:

- Chen et al. (2002b) use ns-2 to model a proposed means of wireless ad-hoc routing choice.
- Dutta et al. (2002) propose pre-filtering as a means of improving the efficiency of simulators, and use ns-2 as an example for testing this.
- Dyer and Boppana (2001) use ns-2 for analysing simulated TCP performance in a wireless network.
- Feldmann et al. (1999) consider the effects of users and networks on the characteristics of IP traffic, and use ns-2 to validate their hypotheses.
- Garetto et al. (2001) use ns-2 in their prototyping of a new form of TCP.
- Hanle and Hofmann (1998) test alternative multicast protocols for the MBONE network using ns-2 for validation.
- In Hofmann et al. (1999), the authors propose improved means of caching streaming media on the Internet, several techniques grouped under an architecture named 'SOCCER' (Self-Organizing Cooperative Caching Architecture). They use ns-2 to prototype this architecture.
- Hu and Johnson (2000) discuss on-demand routing protocols - protocols that only search for the route to a destination node when a sending node sends to the destination. They discuss and simulate means of caching results of the on-demand routing information.
- Kanodia et al. (2001) suggest and prototype (using ns-2 simulation) an intelligent form of QoS module.
- Lan and Heidemann (2003) discuss the system they developed (RAMP, or RApid Model Parametrization) that listens to real traffic, and attempts to parametrise it in order to create near-real-time simulations. RAMP parameters are designed

[1] http://www.isi.edu/nsnam/ns/

to be used with the ns-2 network simulator.
The authors begin by discussing the data sets that they obtained in order to test their program, then go on to discuss the technologies they harnessed to rapidly extract parameters from data. They used wavelet-based time series analysis for scaling, and the Kolmogorov-Smirnoff goodness-of-fit test to determine differences in traffic datasets. They then go on to compare RAMP to another workload simulator, SURGE, showing how RAMP, using fewer assumptions about traffic distributions, more accurately models the patterns.

- Kong et al. (2001) use ns-2 as a simulator to confirm their proposed scalable solution to the security challenge that mobile ad-hoc networks face - the distributing certificate authority functions via threshold secret sharing.
- In Marina and Das (2001), the authors make use of ns-2 to perform comparisons between routing protocols to verify their research.
- Puri et al. (2001) evaluate their protocol for video streaming over the Internet by making use of the ns-2 simulator.
- Rejaie et al. (1999) develop a congestion control mechanism to encourage "TCP-friendliness", simulating it on ns-2 to determine its various properties in a complicated environment.
- Sahu et al. (2000) use ns-2 to validate their research into differentiated services.
- Sinha et al. (2001) use the ns-2 simulator in comparing the performance of two ad hoc routing protocols, DSR and AODV.
- Veres et al. (2000) use ns-2 to simulate the effects of TCP connections on a variety of network setups, with a particular focus on the changes over a period of simulated time.
- Xu et al. (2000) make use of ns-2 to present their proposed algorithms for routing in ad hoc wireless networks where energy is scarce.

Further research which makes use of the ns-2 simulator can be found on their website[2].

2.6.2 SSFNet

Another popular simulator in research, SSFNet is a set of libraries used in programmatic development of simulations. Strong use is made of the 'flow' abstraction to more efficiently model TCP connections.

[2]http://www.isi.edu/nsnam/ns/ns-research.html

The following research makes use of the SSF framework for developing simulators:

- Cowie et al. (1999) discusses the challenges and considerations which are necessary to adequately model the complete Internet without abstraction (which causes a loss in accuracy). Related to Cowie and Liu (1999), both describe solutions as they are applied to the SSFNet simulator. A statement made in Cowie et al. (1999) is another important aspect of network simulation development:

  > "[w]hat works instead is a modeling framework that decouples configuration data from configuration code. A model is built from a hierarchy of self-configurable classes with assistance of a database. The goal at each design stage is to simplify the class code so that it is both verifiably and intuitively correct. If we can verify the pieces, and verify the methodology used to glue the pieces together into a large model, then we can inductively validate even very large, complex models."

  Practically, this means that components in a network simulator should be developed simply, with modularity as a priority. If all simple components in the system are tested and are operating and interoperating properly, then it provides additional validation for larger models using similar components.

- Bai et al. (1999) simulate the effect of two protocols on a wireless network, by observing the corresponding errors that occur.

- Using SSFNet, Briesemeister and Porras (2005) propose and simulate a method for worm detection and countering. They use a broad catching strategy ('group defense strategy') to ascertain the origin and signature of the worm, then rate-limit to slow the spread.

- Li (2001) contrasts the speedup of the SSFNet simulator over that of the ns simulator, a precursor to the current generation ns-2 simulator described in Section 2.6.1.

- Mao et al. (2002) use the SSFNet simulator to test their hypothesis about the effects of sender-side loop detection and withdrawal rate-limiting, proposed features of BGP.

- Nicol (2001) uses a Dartmouth implementation of the C++ API for SSFNet, known as DaSSF. The authors use this simulator to test their composite synchronisation algorithm which searches for conditionally optimal channel assignments.

- Perrone and Nicol (2002) discuss the development of an implemention of the Scalable Simulation Framework (SSF) for TinyOS - an operating system for "smart dust". "Smart dust" refers to the idea of a high number (hundreds to millions) of tiny processing units that can conceptually work together to do large-scale computation.

- Xiang and Zhou (2004) describe ways of defending grids of computers against Distributed Denial-of-Service (or "DDoS") attacks, and use the SSFNet simulation framework for testing.

Other reseach that makes use of this simulator can be found on the SSFNet website[3].

2.7 Other Relevant Readings

Cooperative Association for Internet Data Analysis (2008a) provides a broad set of information on packets captured by a network telescope (or "darknet"), a form of scanning malware countermeasure. This data is useful as it provides insight into the "ambient noise" of traffic on the Internet, and is particularly useful for simulation modeling as it allows for more accurate depictions of traffic from an abstracted "Internet" - allowing simulation development to remain focused without the challenge of modeling and determining realistic Internet traffic.

This data could be used to validate the output of an Internet simulator, but, as explained in Section 7.4, this was not done for this research due to bandwidth constraints.

The ambient Internet noise theme is continued in Pang et al. (2004), where the authors study the "noise" they have acquired through a network telescope and comment on the results they find. This field of research is also investigated by Richter and Irwin (2008), who use a small Internet telescope to make inferences about the composition of the Internet, considering packets that are received and finding specifics about their origins.

2.8 Summary

In this chapter, literature relevant to the development of network simulators was presented. Section 2.2 focused on networking problems and concepts, specifically focusing on the challenge of network structure and design. Section 2.3 considered literature covering Internet and network simulation, citing documents that had made heavy use of simulation techniques and noting those aspects that they made use of, in order to ensure they are included in simulator development.

Section 2.4 began by discussing malware and Internet worms, focusing on Code Red vII (a particularly well-documented case), extracting information about the worms for

[3]http://www.ssfnet.org/publications.html

use in simulation. Section 2.4 then changed the focus to instances where these worms have been simulated, in order to better see the simulation techniques used in this problem domain.

Finally, Section 2.6 discussed other popular network simulators, and cited the large bodies of research that have made use of them.

The concepts taken from this review of current research are used throughout this work: in the process of the design and construction of the simulation software discussed in the following chapter, as well as in the development of worm simulations performed in Chapter 6.

3 Software Design and Construction

3.1 Introduction

Constructing robust network simulators is a challenging, complex task that requires multiple iterations of the implementation and design process. The program evolves as simulations' needs begin to test the boundaries of the capabilities of the simulator.

Because of the evolution of the simulator, documenting the finally produced program is of little use. Instead, the design goals and principles are discussed, and then the initial program is documented. Once this is complete, the challenges that arose in the implementation of the simulations are documented, and the improvements that were made to overcome those challenges are shown.

This chapter discusses the design and construction of a simulation engine, or simulator. Section 3.2 documents the initial design of the simulator, beginning with Section 3.2.1, a discussion of the major conceptual components of the system. Following this, the planned implementation details of the system are covered in Section 3.3, covering a more programmatically-oriented overview.

In Section 3.4, the development of simulations is documented, showing the major forms of simulation components, and discussing how these components interact with the execution system.

In Section 3.6, the challenges faced after developing and running simulations are shown, and the improvements made to the system are documented. Note that the results of the executed simulations can be found in Chapters 5 and 6.

Finally, in Section 3.7, further possible extensions to the simulator are mentioned, with an explanation for their non-inclusion in the current system.

3.2 Construction of the Network Simulator

The network simulator, which was dubbed "GraphSim" for "Graph Simulator", was designed with the ability to develop robust, powerful simulations as the chief priority. Secondary was efficiency, as a major priority was keeping the program executing on a

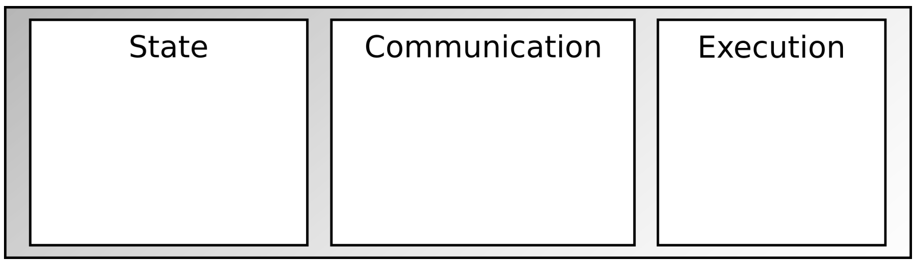

Figure 3.1: Simulator Subcomponents

single system, and simulated accuracy, as trade-offs for efficiency and detail were only made in extreme circumstances.

Design and development of the simulator was performed iteratively - as simulations were found to make more demands of the simulator, the system was redesigned and further developed to extend it's functionality. Because the simulator was created for users to construct their own simulations and components, it was felt that multiple design and development phases would be required, as each generation of simulations would add feature requirements to the simulator engine itself. This resulted in a richer simulator, with tested component sets.

In designing the simulator, two major points of view were considered: a high-level overview which described the system in terms of broad concepts, and a programmatic overview, which took into account the specifics of the programming that would be done to implement the system. This was to ensure that the broad concepts that were initially conceived could be translated easily in the implementation phase of the program, while losing little of the important functionality that was needed when the system was first considered - the goals of this research should not be compromised due to programming challenges that arise.

The initial conceptual design is described in Section 3.2.1, while the pragmatic, programmatic design decisions are detailed in Section 3.3.

3.2.1 Initial Conceptual Design

The initial conceptual design of the simulator centered on the vision of a network of hosts represented by icons, connected with lines, passing packets of information, represented as icons with data on them, around the network. Hosts should be able to be added and connected arbitrarily to the network, and through a simple mechanism, code should be able to be appended to the various components.

Specific forms of nodes (such as an "internet" node, a "router" node or a "host that has been infected with a virus" node) could be made, as could specific forms of connection ("10baseT" connection, "lossy" connection, or 'perfect' connection) and packet ("UDP" packet, "worm-containing" packet). These could be modeled within the simulator,

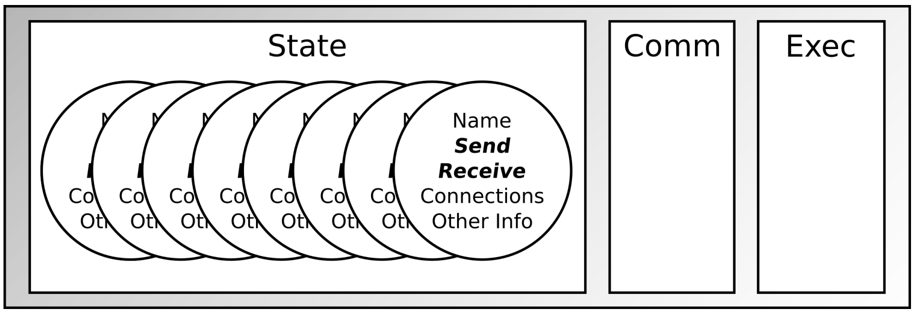

Figure 3.2: State Component of Simulator

either as subclasses of nodes (if their behaviours are consistently different within the bounds of a simulation) or by adding additional attributes.

This novel system was designed to allow for very rapid conceptual design of networks and simulations, with simple components that related strongly to their real-world equivalents by allowing attributes of any form to be assigned to them. For instance, a packet component with these attributes could have a "viral payload" attribute, set to "blaster worm". With properly specified attributes in an appropriately designed host (such as an "infectable" host, with an "act upon payload arrival" attribute), this could have semantic ramifications for the remainder of the simulation (such as causing more infected packets to be generated from the host).

The simulator was conceptually divided into three subcomponents, as visualised in Figure 3.1: the state system to hold nodes (considered in Section 3.2.1), a communication system to model connections (considered in Section 3.2.1) and an execution system which would act upon the previous two components (considered in Section 3.2.1).

State System

The conceptual design of the state system was a series of nodes with a variety of attributes and behaviours, as visualised in Figure 3.2. Following the robust theme of the simulator, they must be capable of simulating the behaviours of a very large range of devices. Any device that could be connected to any sort of network should be able to be represented.

Because of this, a very simple and broad set of minimum requirements for node implementation would have to be defined. Upon reflection, the following behaviours and attributes were considered to be of primary importance, and thus were added to the template for nodes:

- A unique reference or name
- A behaviour that could, under certain circumstances, allow the node to send something (presumably a message of some sort) from itself to another node via

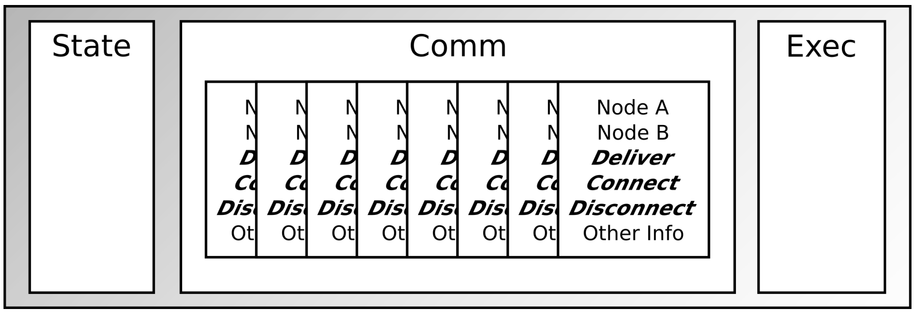

Figure 3.3: Communication Component of Simulator

a connection

- A behaviour that could, under certain circumstances, allow the node to receive something (presumably a message of some sort) from another node via a connection
- A list of connections that are connected to this node
- An enlargeable set of secondary, "semantic" attributes that contained a semantic descriptor (such as "IP Address") and its associated piece of information ("192.168.0.1")

With this template in place, programming of the node template could begin. The "node module", described in Section 3.5, was the implementation of this template.

Communication System

The conceptual design of the communication system was a series of connections between nodes. The isolation of the connection system and the state system was important, as the connections between nodes can have many more attributes than just the unique addresses of each node. Concepts such as bandwidth, traffic and lossiness must also be representable in the communication system.

In order to generate a template for a connection, much like that used in the state system for nodes, it was necessary to set minimum requirements. The following behaviours and attributes were considered of primary importance, and used in the connection template, visualised in Figure 3.3:

- A reference to the nodes that the connection connects
- A behaviour that takes a message from one node, and delivers it to another node
- A behaviour that can disconnect the connection from the current pair of nodes, and connect it to another pair

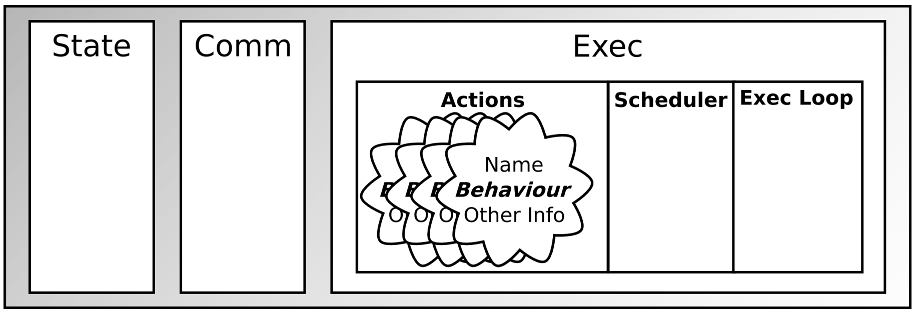

Figure 3.4: Execution Component of Simulator (Action Subcomponent)

- An enlargeable set of secondary, "semantic" attributes that contained a semantic descriptor (such as "current lossiness") and its associated piece of information ("21%")

With this template in place, programming of the connection (or "conn") template could begin. The "connection module", descibed in Section 3.5, was the implementation of this template.

Another important conceptual component of the communication system was the messages that would be passed using the connections. The messages themselves were rather simple, and comprised of a "header" (not to be confused with protocol-specific headers that would be part of the packets that these messages represent) and a "payload". The "header" contained the unique node names of the sender and recipient of the packet. The "payload" contained the information that the message itself would contain (it would be in this section that a protocol-specific header would be found were this concept to be implemented).

In keeping with the robust nature of the simulator, it was also decided that the enlargeable set of secondary attributes were added to the message system, as various semantic concepts could be associated with them. Protocol, fragmentation and other attributes could then easily be associated with the messages in the system.

Execution System

The conceptual design of the execution system is more complex than the state and communication system. It was designed with three major subcomponents: action components, the scheduler component, and the main execution loop.

Action components This component was, in fact, defined as a series of mini-components that could be combined or altered to perform operations upon the state and the communication systems, visualised in Figure 3.4. Through this manipulation, the system could be advanced from its initial states.

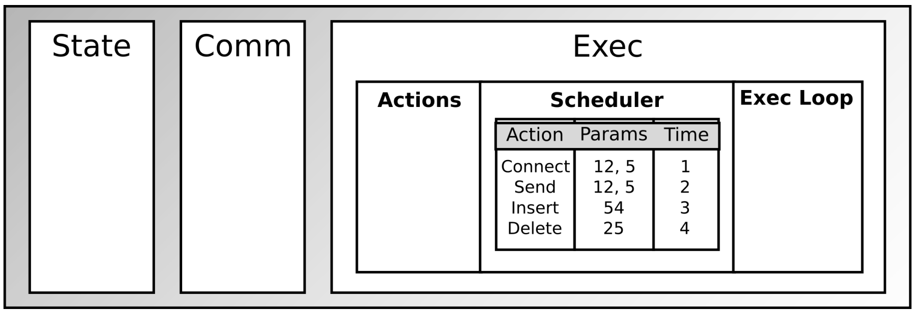

Figure 3.5: Execution Component of Simulator (Scheduler Subcomponent)

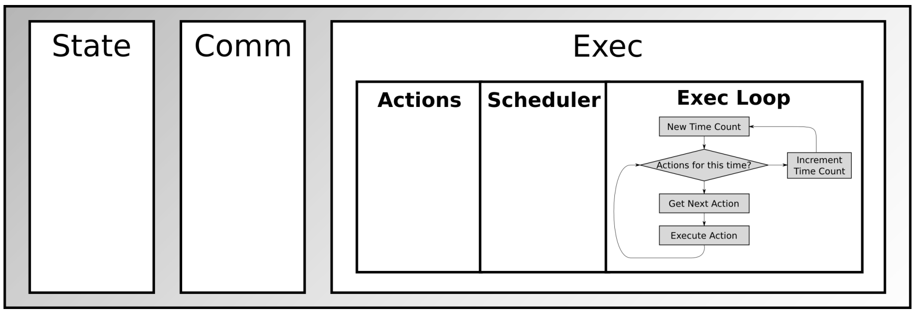

Figure 3.6: Execution Component of Simulator (Main Execution Loop)

Scheduler component This takes the form of a table (shown in Figure 3.5), combining action components with specific parameters (such as which part of the system to operate upon), and an associated simulated time. This table could be manipulated (by action components) at any point in the simulation, so that single complex actions (such as "send a packet via four hops") could be unpacked into several simpler actions (such as "send a packet from one node to another", for each hop).

Main execution loop An execution loop contained a "timer", which would increment upon completion of the loop. During the loop, each action component that was associated with the newly set timer in the scheduler would be implemented, with the associated parameters set, and the associated code executed. Once complete, the component would be disposed of. This can be seen as a flowchart, shown in Figure 3.6, and magnified in Figure 3.7.

The only varying part of this system was the action component, so a template would have to be generated for it. In order to generate a template for an action, much like that used in the state system for nodes and the communication system for connections and messages, it was necessary to find minimum requirements. The following behaviours and attributes were considered of primary importance, and used in the action template:

- A unique name that should be used as an identifier in the scheduler's table.

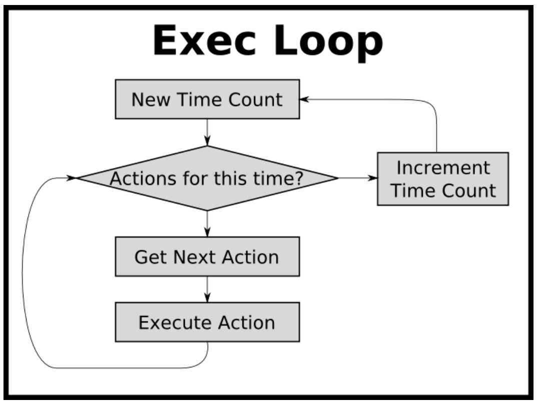

Figure 3.7: Main Execution Loop

- A behaviour that will execute a series of commands specific to the form of action.
- A section of code to be executed when the main execution loop reaches the action component, which may or may not take parameters.

This template is implemented in Section 3.5.

3.3 Programmatic Design

Having completed the conceptual design, more specific details of the system needed to be outlined. Expected core and complex components of the simulator had to be planned and detailed, in order to avoid problems arising later in the implementation.

These components were considered to be the following:

The container component The container component is largely the parent component of the rest of the system.

The scheduler component The scheduler component, a sub-component of the engine, is conceptually dealt with in Section 3.2.1.

The execution engine component The execution engine component uses the scheduler component to execute actions. It is also a sub-component of the engine component, conceptually detailed in Section 3.2.1.

The plugin management component This component is the primary component of interest in this research: the conceptual templates detailed in all of the sections above are instantiated as modules and used to represent the network that is to be simulated.

Container Component

The container is a wrapper component for the entire simulator system. It holds all of the modeled components of the simulator, and is the part that "represents" those objects that the simulations are describing, at any single given moment in time. While it may be wrapped in a GUI and use external files for inputs and outputs, for the purposes of execution, the engine contains all of the components necessary to execute the simulations.

All of the programmatically detailed components below are sub-components of the container component. The scheduler and execution engine are part of the main execution loop of the program, and operate upon the simulated network, comprised of modules that have been imported via the plugin management component.

The container component was designed to begin its execution by initialising the system for simulation. This procedure involves instantiating a "setup" module that will create the structure of the network (comprised of nodes and connections, with the initial messages queued to be sent), and the major actions that are to take place. It should then add this module as an action component to the scheduler as the first action to take place. Once that is done, the execution engine should then be started. After this, the main role of the engine is to perform background logging of the state system, keeping a record of the current state of the system periodically. This serves two purposes: firstly, it allows for system restoration at the logged point if the logs are extensive, and secondly, it allows for analysis of the current state of the system - initially for debugging purposes, but finally in order to gather information about the state of the system at a specific time.

Scheduler Component

The scheduler component is a sub-component of the engine component. It contains an ordered table (implemented as a two-dimensional array) with three attributes: a time, the name of the action component to be executed at that time, and optionally a series of parameters that are associated with the action at that particular time. An example of an action that would be without parameters would be a "finish execution and terminate simulation" action, while an example of one which would include parameters would be a "send a message from host X", where X could be specified as a parameter, as could the contents of the message if so desired.

The scheduler has to be globally accessible, as other action components are required to alter it. A common example of this was the sending of a message to a host that was not immediately connected to the sending host. A simple action that was in the scheduler, such as "send a message from X to Z", where X and Z are separated by host

Y, could be interpreted and re-added to the scheduler as two actions: "send a message from X to Y to arrive at time 2" at time 1, and "send a message from Y to Z to arrive at time 3" at time 2.

The scheduler itself does not execute these actions. That is the responsibility of the execution engine. The scheduler is merely a complex means of storing the order of actions to be executed.

Execution Engine Component

The execution engine component is a sub-component of the engine component. It operates upon the scheduler component, by requesting actions that need to be performed at specified times. When the scheduler returns these actions, along with any associated parameters, the execution engine begins to instantiate these action components and execute the code associated with them.

The execution engine itself does not contain the code that is executed. The code is part of the action component that it receives from the scheduler. The action components are written as seperate action plugins, which are detailed further below, in 3.5 and in the next described component.

Plugin Management Component

The plugin management component is the final sub-component of the engine component. In order to properly understand this component, it is first necessary to understand GraphSim plugins:

The plugins that are used by GraphSim correspond to the four types of templates detailed above in Section 3.2.1: node plugins, connection plugins, "message", or packet plugins, and action plugins. Each of these plugins defines a broad form of the requisite type. A node plugin, for instance, could be an "infectable host" plugin, or an "IPv4 host" plugin.

The plugin manager, then, is the part of the engine that governs the use of the plugins, containing information about plugins, and allows for their instantiation and use. As such, it is the most important part of the simulator.

It is broken down further into the following parts:

Plugin Location, Parsing and Import As plugins are stored externally to the simulator, they must be located on the disk, parsed for correctness and imported into the simulator.

Plugin Instantiation Once plugins have been imported into the simulator, they must be instantiated on an on-demand basis.

Through the use of these two parts, the plugin manager can import and instantiate new plugins as needed.

Robustness in Simulation

Robustness is an important feature in any form of simulation that can easily be adapted to suit a researcher's needs. It can be defined as the degree of capability a system has to represent or model reality.

As a result, simulation robustness can be implemented in two ways:

1. Creating no framework at all, as a framework imposes conceptual limitations
2. Creating a very generic framework, using the minimum possible restrictions on modeling while still imposing order on the system

While the first option is tempting, it leads to difficulties in development. With no framework in place, each component needs to be uniquely developed for every simulation, and the simulator itself will need continual modification to support the new components.

The second option, while not "absolutely" robust (in that it imposes some order on the system) can still allow for streamlined development while allowing rich representation of objects in simulation.

3.4 Construction of Network Simulations

Simulations in GraphSim are constructed in the order and style shown in this section. They would primarily be composed of instantiated plugins that were introduced in Section 3.3. Though it is constantly being updated and improved, the simulations described here will explain the modules and plugins with the underlying assumption that it is based on the build of the execution engine that was current in late 2007.

The modules that are used for the simulations derive from one of four types: nodes, connections, packets and actions.

The decision to do this comes from the notion of templatised constructs mentioned in Section 3.2.1. These templates were used to define the final implementation of the interface and abstract classes that formed the foundation of the modules.

In order to allow for the broadest range of userbase for simulation development, the plugins were designed for use with Microsoft's CLR (Common Language Runtime), so that any language that has been prepared for CLR use (such as Python, C or C++) can be used for writing simulations. Testing and preparation was done in C# on the mono platfrom, so the specific references shown below are tailored with this in mind.

3.4.1 Planning

Before any simulation is created, careful planning needs to take place. The minimum information required before programming should begin is as follows:

- What sorts of hosts will there be in the simulation? Should these be placed in a broad category (using a single plugin) with specialised attributes, or are they diverse enough to justify several plugins?
- How are the hosts connected, and how would the user wish the connections to be simulated? Does the user want packets to spend time 'on the wire', or would they prefer immediate packet delivery? Are all hosts mesh-connected, or do they follow some network structure?
- How complex does the user wish their packet plugins to become? Because several million packets may be simulated simultaneously, efficiency can quickly become a consideration. Does the user want the 'payload' to accurately represent the contents of a packet, or are they prepared to allow for high-level abstract messages to be sent?
- Which classes of action occur in the simulation? Does the user need to make an action plugin for each action in the system, or can they be refactored so that it uses fewer actions, with parameters passed that affect their behaviour?

Once these questions have been answered, the user should have a clear knowledge of which plugins should exist in the simulation, and the complexity at which these plugins will be modeled.

3.5 Modules

Because of the robust nature of the simulation, all plugins contain an enlargeable data structure, used for adding semantic content to the simulated components as needed. For ease of use, consider that the structure operates as a dictionary with strings used as keys and strings used as values, that grows like a vector. The initial design work termed the keys to the dictionary 'tags', so the data structure is termed a 'tag list'.

Each of the module types throughout this section have diagrams to explain the structures that they use. These are included for clarity and explanation of the construction choices that were made when fundamentally defining the components considered as building-blocks for a simulator.

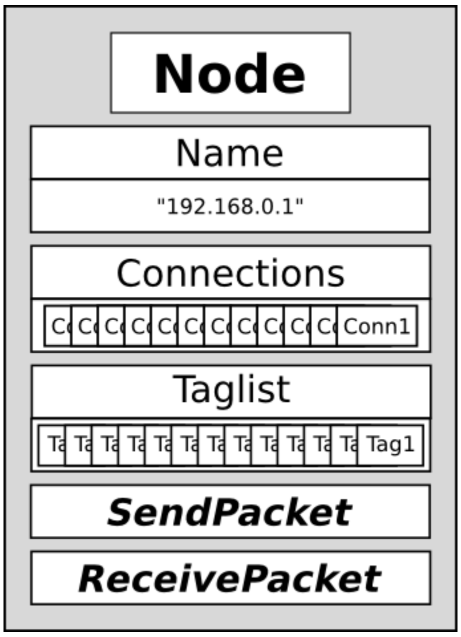

Figure 3.8: Node Module Diagram

Node Modules

Node modules, as mentioned in Section 3.2.1, need a unique reference, or name, a sending behaviour, a receiving behaviour, a list of connections, and a tag list. This could be modeled using data structures like those shown Figure 3.8.

This header could be used in the interface for node objects. When the interface is implemented as, say, a 'infectable host' node, then it would override the header's basic attributes and behaviours with it's own.

Connection Modules

Connection modules, as mentioned in Section 3.2.1, need a link to the nodes they connect, a behaviour that corresponds with the sending and receiving behaviours of the nodes, connecting and disconnecting behaviour, and a tag list. This could be modeled using data structures like those shown in Figure 3.9.

This header could be used in the interface for connection objects. When the interface is implemented as, say, a 'high traffic' connection, then it would override the header's basic attributes and behaviours with its own.

Packet Modules

Packet modules, as mentioned in Section 3.2.1, need information about origin and destination, a payload, and a tag list. As can be seen, packets are very simple data

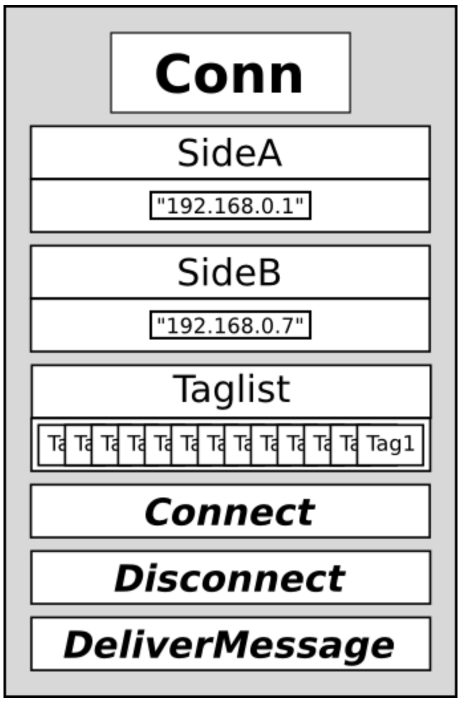

Figure 3.9: Connection Module Diagram

structures that contain no behaviours of their own: they are sent over connections, and nodes can interpret their payloads on arrival.

This packet interface, then, could be modeled using data structures like those shown in Figure 3.10.

This header could be used in the interface for packet objects. When the interface is implemented as, say, a ‘udp’ packet, then it would override the header’s basic attributes and behaviours with its own.

Action Modules

Action modules, as mentioned in Section 3.2.1, need a unique reference, or name for the scheduling table, a behaviour to be executed when their time of execution is reached, and a tag list. Because the execution behaviour can optionally take parameters, using C#’s overriding mechanism it is possible to define more than one function with the same name, one to be called if the parameter list is included, the other if it is not. This could be modeled using data structures like those shown in Figure 3.11.

This header could be used in the interface for action objects. When the interface is implemented as, say, a ‘node sending’ action, then it would override the header’s basic attributes and behaviours with its own.

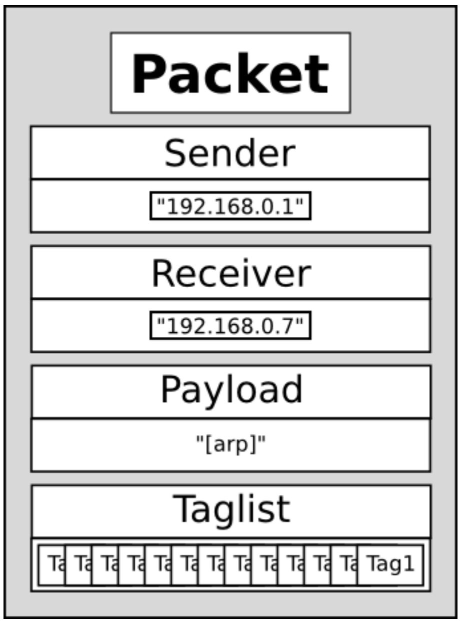

Figure 3.10: Packet Module Diagram

3.5.1 Execution Setup

When the engine component is initialised, it goes through a specific sequence of function calls. Optimally, every new simulation that is run should require a minimum of codebase alteration, so it is in the engine that robust initialisation should occur.

In GraphSim, this is done by specifying which simulation should be run through a 'setup' action plugin, which is executed immediately upon simulation startup. Each simulation has its own setup plugin, and the setup plugin is the only part of the engine that changes between simulations.

The setup plugin is broken into two sub-functions: setting up the network structure, and setting up the events that should occur throughout the simulation. Both are largely dependant on the simulation to be run (for instance, a simple network testing simulation which has a very simple structure setup, but might have very detailed event scheduling for micromanagement, while a large-scale full internet simulation might be quite the opposite). The network structure typically runs through a loop, creating, naming and adding instantiated nodes to the network structure. The event setup typically starts with the sending of packets from various hosts.

In most of the more advanced simulations that have been built with GraphSim, nodes are created with outgoing and incoming packet 'queues', so at setup time nodes are initialised, and packets are then enqueued to various nodes. In the event scheduling system, a 'send packet' action plugin is added to the scheduler with the name of a specific node that has enqueued packets passed as a parameter.

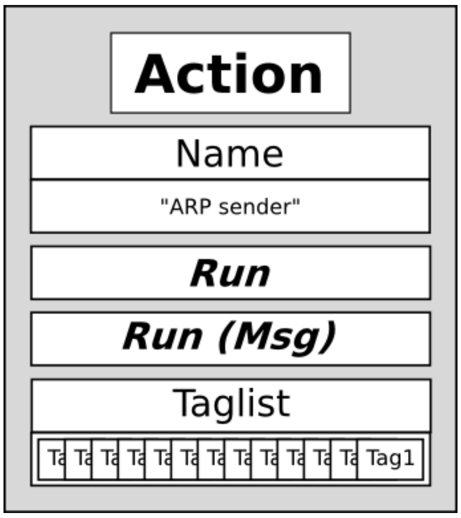

Figure 3.11: Action Module Diagram

3.5.2 Simulation Development

In this section, an example of a simulation that could be set up and executed is described.

The first components developed in the simulation are typically the “start” and “finish” action components. The start (or setup) component typically instantiates the node-, connection-, and several packet-objects, and will (depending on the simulation) also schedule the actions that are to be performed throughout the simulation.

The finish action will typically stop any other actions executing, then print a log of the current state of the system. It will then close the execution engine and stop the simulator.

Once this has taken place, development will typically start on the nodes and connections that will be used in the simulation - at a design level, the attributes and behaviours will be considered, then implemented as “tags” and methods respectively. If the behaviour of the node will be significantly different from that of other node types, then an entirely new class of node will be created to adapt for that. Similarly, the connection will have variables and methods attached.

Packet objects typically do not require any significant changes, as the range of behaviours that a packet could represent that would have an impact on a system, is limited.

Finally, the actions that would take place that would be modeled. The way in which these are modeled will typically be prototyped and revised several times in the course of a simulation’s development, as action components in particular require iterative development. Common challenges (raised in Section 3.6.2 below) are easily aggravated by any bugs in the action component code.

While action components require a particular attention to detail, the entire simulation development process is iterative - each pass further adding detail or removing unnecessary elements of the simulation. Once the simulation developer is satisfied that the simulation is complete, then the simulation is done.

3.5.3 Example Simulation Development

The most complex simulation considered in Chapter 5 is a mixture of two types of nodes, in a mesh-connected network, that connects to a further type of node, and is capable of routing.

Setup and Finish Components

The setup action will need to create a network of nodes (developed later but noted during the design stage) which are mesh-connected with connection objects, and which send a large number of randomly addressed packets (some to the local simulated nodes and some to randomly generated addresses).

The finish action will log the number of packets received by each IP address, addressing each node in turn (all node types that have been simulated) and requesting the number of packets they have received and (where applicable) which IP address they were addressed to. This should then be printed to the screen, allowing the user to see the outcome of the simulation.

Node Development

The three forms of node that this simulation will require can be referred to as “nodes”, “routers” and “network nodes”. Each will require a specific set of attributes to represent them.

The network node will need to store information on which abstracted nodes have received packets. Router nodes will require a “routing table” attribute that stores information about the types of nodes to which they might route packets.Nodes (as well as router nodes) will need addressing information so that packets may be routed to them.

Behaviours will also be different - router nodes must route packets when they arrive (unless the packet is addressed to the router node), while “normal” nodes should indicate that a packet has arrived in their queue. Network nodes must interpret any incoming packets and indicate to which abstracted node the packet would be delivered.

Connection Development

As this is a simple simulation with no special note being taken of bandwidth or traffic, a simple connection can be modeled - the connection will simply dequeue a packet from the "front" of a connected node's outgoing queue and enqueue it on the receiving connected node's incoming queue.

Packet Development

Packet objects also remain largely unchanged from the simple case - addresses of sender, receiver, and a string for a payload suffice for this simulation.

Action Development

Two action types (other than the setup and finish actions) require development for this simulation - the sending action (which will execute the send behaviour of the connection objects where appropriate) and an enqueueing action which will generate a variety of random packets and place them on nodes' outgoing queues. A side-effect of the enqueueing action would be to instantiate send actions for each packet which is enqueued, in order to activate the sending effect.

3.6 Challenges and Evolution

In building a simulator of any significant scale, three major challenges arise. The massive amount of memory required to hold the information pertaining to the state components becomes untenable on a single computer as the number of nodes and level of detail rise. This is aggravated by typically poor means of associating semantic information with nodes. Finally, access speed is an optimisation challenge, due to the massive number of hosts and the need to rapidly acquire a specific host upon which to operate.

3.6.1 Computation Concerns

Li (2001) states that: "The major difficulty in simulating large networks at the packet level is the enormous computational power needed to execute all events that packets undergo the network". The proposed simulator does not consider computational power a shortcoming: instead of modeling every packet to simulate traffic, an attribute can be attached to a connection component (See Section 3.5).

This also saves memory, a scarce resource which forms the basis of several challenges mentioned in Section 3.6.2, by removing the necessity of containing every packet in memory and using an abstraction of "traffic" instead.

3.6.2 Challenges

Before any attempts at solving problems can be made, it is imperative that the problems that are being addressed are outlined comprehensively and documented, in order to ensure that they are given proper attention, are fully understood, and are solved in isolation, as a novel solution to one problem may not be useful for development of similar simulators on systems different to those used in this work.

Address Space

Memory shortage is a great challenge in a simulation system. If we were to try to simulate the entire Internet, we would require $2^{32} = 4294967296$ hosts to be simulated (Removing Class D and E networks, it still numbers approximately four billion addresses). Paxson and Floyd (1997) comment on this in their paper, "Why We Don't Know How to Simulate the Internet".

If we consider the amount of memory required to hold a pointer to a node structure, we will find that the simulation will require $2^{32} \times 2^5 = 2^{37} = 137438953472$ or 128 gigabytes of memory. This is not perfectly accurate, as special networks, such as class D and E networks which will not have to be simulated, have been ignored. However, these networks are relatively small, and the amount of memory used to represent them would be insignificant - the amount of memory needed is still vastly greater than most modern desktop computers can hold. This does not include information about the hosts, this is merely the memory used to hold pointers to all of them. This also does not include the associated communication system which grows exponentially as we add hosts to the system (assuming the system is mesh connected).

Unnecessary Memory Use

As the simulator becomes more robust, the hosts are expected to hold more and more information. If we are allowing several different concepts to be represented in our nodes, then they can quite easily hold dozens of variables defining operating system, hardware specifications, etc. As shown in Section 3.6.2, the number of hosts in the system might be large, and the memory structures used to hold them explode the amount of memory used. The challenge of holding large quantities of information in a very large number of structures is in optimising the node detail access, and the solution is presented in Section 3.6.3.

Node Access Time

Most modern programming languages use a simple data structure such as a ‘Vector’, or ‘List’. It would seem tempting to use a structure like this to store the large number of hosts in our system, as it is a standard in the language, as well as straight-forward. The challenge arises, however, in the means of access.

An optimisation is possible if we assume that we are simulating a series of nodes from an IPv4 network. In order to extract a node from the data structure given the IP address in some form, we need to traverse the structure, comparing every element with the associated IP address (assuming that the IP address is stored as a detail in the node).

This is barely noticeable in trivial simulation examples, but when the simulation grows large, at every time tick the simulator would have to search through several million elements in the array, thousands of times. The access time for finding a node must be incredibly quick in order to facilitate rapid simulation.

3.6.3 Solutions

Having considered the problems of massive address space, unnecessary host memory use and node access time, it is necessary to document the solutions that were proposed and show those recommended for simulator development.

Access optimisations

In order to solve the access time challenge presented in Section 3.6.2, it is necessary to use optimised methods of searching for an object in memory. By using data structures that are ordered, we can improve search times significantly. Presented here are the two recommended methods: trees and hash tables.

Trees Trees can be used to traverse the IP space very rapidly and efficiently. By separating the hosts by IP address into their hexadecimal pairs, we create a tree that is four levels deep, and closely approximates network structures. It is also easier to optimise for space, detailed in Section 3.6.3. Because the time required to find a node is constant (four traversals), efficiency in finding nodes is markedly improved.

Hash Tables Hash tables can be used for even faster node retrieval. While trees require four traversals, hash tables can immediately return the node. The only processing in order to access the node is to apply the hashing function to the IP address of the host.

In terms of memory, hash tables are slightly superior to trees - trees require a logical link to be kept between related elements, while hash tables use keys (which conceptually are not stored in memory, but which are hashed at runtime) for memory access.

Space optimisations

A solution to the problem that a shortage of memory presents is to only instantiate nodes and create pointers to them when they are required. When the simulation is started, the Internet can be described as a single entity. As a specific host (say, 146.231.115.89) is addressed, a new node can be dynamically created to represent the host. At runtime, then, the host can be instantiated and detailed using statistics and randomly generated values to represent its attributes. A greater challenge exists if the user wishes to use every host on the Internet (or a particularly large network) for their simulation. In this case, it is possible to do manual page swapping to a hard-drive, though access time will be much slower than if the simulation were to be kept in primary memory. Another alternative is heavy reliance on detail optimisation, detailed below.

Detail optimisations

The solution to the challenge of host memory use, is to optimise the way in which data is stored in memory. This can be done in three ways: by altering the way in which variables are being stored in memory, by using dynamic data structures that only use memory when required, and by creating reverse detail lists.

Efficient Storage When designing nodes, this factor should be taken into account. Using large, memory inefficient data structures for these details will result in a large expenditure of memory. Avoid the use of strings and other list-based structures if possible, and prefer integers, enumerations and Boolean values. In places where strings are required, determine if it is possible to use a hash-function and use a lookup table.

Dynamic Data Structures It is preferable, when creating nodes, to assign no details to them, and create a dynamic data structure that can hold information that shows differences between the node and the norm. In the case of a simulation where most nodes are heterogeneous, this may present problems. If every node differs from the norm, or if the number of details that must be represented are few, then the overhead of a dynamic data structure may overshadow the saved memory. In this case, it is better to specify details for every node statically.

Reverse Detail Lists When the number of nodes that have a specific detail is small, or, more importantly, if some action needs to performed on all nodes with a certain property, then reverse lists become a necessity. A reverse detail list stores a list of all hosts for which a detail is pertinent. An example of its use is in Internet worm simulation: a list storing all nodes that have been infected by the worm (and updated every time the worm infects a further host) has significant benefits over storing the details of the infection in the node itself. The first of these benefits is greatly improved efficiency. In order to find all the nodes in the system that are infected with a worm, it would be necessary to visit every node and determine its infection status. With reverse lists, it is a simple matter to traverse the list and act upon each entry. The exchange for efficiency in reverse detail lists is the extra memory required. The overhead of lists for details may be greater than the amount of extra memory that would be used to hold those details, especially if dynamic data structures (mentioned above) are used.

3.6.4 Infection Simulation

One special case of a programmatic/representational challenge that would be encountered in the development of a network simulator is that of infection simulation. Because Internet worms can spread so rapidly (the Warhol and Flash worms described in Nazario (2003) can spread globally in minutes and seconds respectively), the challenge of representing this in memory arises.

Two options are available for simulation of infection:

Infection lists store the lists of infected hosts on a global level

Host parameters keep the infection information on the specific simulated node

While the latter option is tempting (as the specifics for each particular infection can then be observed, such as duration of infection and number of re-infection attempts), it introduces a great challenge in the worm propagation mechanism: having to find each node that has been infected then becomes much more difficult. Infection lists are a simpler solution - a simple loop can run through a list of infected hosts, placing new infection packets on each hosts' outgoing queue and sending them.

While a hybrid option is also viable (and allows for greater infection details to be recorded while still allowing for simplicity of execution), it results in large redundancies in memory, which would be the scarcest resource in the simulation.

3.7 Further Extensions

As stated in the introduction to this chapter, projects such as GraphSim are continually evolving to allow for broader simulations to be run. This has been complemented by the plugin system that allows for incremental improvements to simulations via plugin evolution, so that the entire system does not need to be changed in order to facilitate improved simulations.

However the system still has several areas of improvement that could be implemented, but due to constraints of brevity and scope, they are left as extensions. They are listed here, and discussed.

3.7.1 Real-Time Visualization Integration

GraphSim was initially conceived as having a rich Graphical User Interface (GUI) with which the user could interact with the simulated network. This would allow for a visualisation of simulations initially, and once development matured, eventually designing of networks at a GUI level.

While it would make an excellent extension to the system, it was considered a secondary priority. Visualisation of network simulations can be done as post-processing, and does not need to be a core component of the system - especially if logs of simulated activities are kept, so that a visualisation tool can show the states of the system.

3.7.2 Real-Time Parameter Alteration

Because GraphSim operates with discrete time and scheduling, it should be possible to "pause" execution of the simulation. In this temporarily halted state, parameters in the system could be adjusted, should they need alteration.

Theoretically, this could be done using plugins and user input, however it was not considered to be of enough use that it should be included as a core part of the system. It is assumed that a simulation can be set up in an initial state and thereafter, no further user interaction will be necessary. All events that occur after the start of the system can be added at the beginning as scheduled events.

3.8 Summary

The planning and construction phase of simulator development is an important part of the development process. By considering the development options before writing the program, it becomes possible to determine where programming challenges and problems will arise.

In Section 3.2, simulator development is discussed. First the concepts behind simulator development are considered, then followed by the concerns that apply to the programming phase.

Conceptually, the simulator needs several core components: a system to retain the states of the various simulated elements, a system to allow communication between these elements, and a means of executing changes upon these elements.

Programmatically, the simulator will also require a variety of systems: an execution engine that will hold the simulation's information as well as the instructions for changing them, a means of representing the passing of time in an ordered fashion, a system for applying changes to components in the engine, and a means of loading and using an array of modules that each represent some element of the real-world, used in simulation.

In Section 3.4, the development of simulations is considered. The various types of modules that will be used for simulation development are considered (nodes, connections, packets and actions) and detailed.

The anticipated development challenges are then stated, and solutions to these problems are then proposed.

Finally, possible extensions to the proposed system are mentioned. These extensions are all possible means of enhancing the system but do not contribute to the core functioning of the system, but are mentioned for readers who may wish to do further research in this subject field. Appendix C contains a complete list of the plugins developed during the course of this research.

4 Simulation Design Decisions

4.1 Introduction

In order to validate the proposed simulator's ability to accurately and easily simulate real-world scenarios, a series of simulations should be run that can be tested against observed data. Many problem domains may be simulated, however in order to simplify these simulations, a single problem-domain was heavily explored - the simulation of internet worms. The justification for this choice of subject can be found in Section 4.2.3.

Section 4.2.1 also describes and defines many of the subjects of the simulations. It introduces malware as a core focus of the research, and provides a taxonomy of the forms of malware used throughout this document. It then goes on to justify the use of Internet worms as a subject of simulation.

Section 4.2.4 continues the discussion of simulation subjects by introducing malware countermeasures, such as network telescopes and counter-worms.

In Section 4.3, the initial testing and calibration simulations are discussed. Section 4.3.2, specifically, deals with the various versions of plugins that were released, covering the capabilities that were added or removed, and stating the justifications for these decisions.

4.2 Simulation Focus: Malware

Malware is a popular source of technology research, due to the importance of information security in modern networking and the Internet. Simulation is a tool used in research for the development and evaluation of scenarios, and as such, plays an important role in malware research. An example of simulation use for research is the ISEAGE project, described in Iowa State University Information Assurance Center (2008). Furthermore, because "live" malware is dangerous for use in research, simulation plays an even more vital role by allowing security researchers complete safety while still studying the effects, causes and possible countermeasures to malware.

In this section, malware (particularly the form of malware known as "Internet worms") is defined, evaluated and considered as a subject for simulation in research.

4.2.1 Malware as an Internet Phenomenon

Malware is defined in the Oxford English Dictionary Online (2008) as "Programs written with the intent of being disruptive or damaging to (the user of) a computer or other electronic device; viruses, worms, spyware, etc., collectively."

Malware is a collective term for all software that has been written with malicious intent. The growth in malware presence on the Internet has resulted in the field of computer security growing tremendously, as personal and organisational protection against malware becomes a necessity.

4.2.2 Malware Taxonomy

Malware can be grouped into several specific classes, depending on intent, function and propogation method. The three main types that are of interest to this work are *virii*, *worms*, and *trojan horses*.

Virii Virii (singular: virus) are defined in Oxford English Dictionary Online (2008) as "a program or piece of code which when executed causes itself to be copied into other locations, and which is therefore capable of propagating itself within the memory of a computer or across a network, usually with deleterious results [... especially] one capable of being inserted in other programs". This definition is broad enough to incorporate worms, mentioned below. If we remove worms from this definition, then virii can be said to be non-self-propogating malware, which spread via human intervention (whether conscious or unconscious), but which deliver the malicious payload autonomously.
Examples of classic virii include the "Stone" (See F-Secure Corporation (2008)) and "Michelangelo" (See Computer Emergency Response Team (1992)) virii.

Worms Worms are defined in the Oxford English Dictionary Online (2008) as, "a program designed to sabotage a computer or computer network [... especially] a self-duplicating program which can operate without becoming incorporated into another program." This definition is limited to malware with autonomous spreading mechanisms, and autonomous malicious payload delivery, even though some worms can later change mode in order to allow malicious users into the system, changing to a Trojan Horse (see below).
Popular examples of worms include the "Blaster" worm (see Dougherty et al. (2003)), "Welchia" worm (see Perriot (2008)), and "Code Red" worm (see Danyliw and Householder (2001)).

Trojan Horses Trojan Horses, or simply Trojans, are a class of malware that insinuates itself into a computer system or network (via human or autonomous means), and then proceeds to 'open a back door', allowing further illegitimate access to the system to other entities, be they malware or user. These are considered a different class to those previously mentioned as the malicious payload is not delivered by the malware itself: it merely allows other, unauthorised users of the system to perform those malicious acts.

For purposes of this research, specific attention has been paid to Internet worms.

4.2.3 Rationale for Worm Simulation

Worm simulation was selected for the subject of simulation testing. There are several reasons for selecting worm simulation as an initial and suitable subject for framework testing:

Largely Homogenous Activity Internet worms operate by reproducing themselves in a largely homogenous manner (excluding polymorphous worms, which are outside of the scope of the testing - Nazario (2003) points out that the Ramen and Nimda worms displayed multiple attack vectors, the first component of polymorphic worms). Because each similar infection acts in a largely similar way to all other infections, it means that the range of activities to simulate is kept small, allowing for simpler simulations.
This will reduce the number of action components (See Section 3.4) that would need to be developed.

Scaling Activity Density Worms initially begin with very few activities to simulate, and rapidly scale to massive amounts of simulatable network activity. This allows for a broad range of interesting simulations, scaling from single processor, nearly immediate simulations, to the possibility of grid-based simulations, run over hours, days or weeks.

Internet Scale Worms are an Internet-sized event. By simulating Internet worms, massive simulations become the norm, forcing development to keep efficiency (both in terms of computation and memory) as a high priority.

Well Documented Several famous worms have been dissected, behaviourally and structurally analysed, and documented extensively. With a large quantity of available documentation, simulations can remain relevent to the field of security research - Internet worm propagation or scanning algorithms can be accurately depicted using the scanning algorithms used in the worms themselves.
The worms modeled in this research include:

- The Witty worm, documented in Stewart (2004) and Shannon and Moore (2007).
- The Code Red vII worm, documented in Zou et al. (2002), Wagner et al. (2003) and further in Section 2.5.2.
- The Blaster worm, documented in Bailey et al. (2005) and Castaneda et al. (2004).
- The Welchia worm, documented in Castaneda et al. (2004) and Perriot (2008).

Simulations Useful to the Security Research Community By extending the simulations into areas that are currently only being considered or prototyped (such as Network Telescopes, Moore (2002) Moore et al. (2004) and Inter-Network Containment, Coull and Szymanski (2007)), the results of simulations can be useful to the security research community for prototyping, testing, and development.

4.2.4 Worm Detection and Defence Tools

Worm detection and defence are key parts of Internet security. Through early detection systems, complex worms can immediately be addressed and countered, while simpler worms can be defended against using automated tools, without the need for (comparatively slow) human intervention.

In this section, three tools are mentioned - network telescopes (a form of detection), tarpits (a form of defence) and helpful worms (a controversial form of aggressive worm defence).

Network Telescopes

Network telescopes, sometimes termed “darknets”, are a form of random scanning worm defence. Network telescopes work by listening on IP addresses that are not published (such as in DNS) other than for routing, and which run no valid services, where no standard traffic is expected. By not publishing the IP addresses, one can safely assume that the incoming traffic is not valid network traffic - it will either be from misconfigured applications (such as incorrectly entered IP addresses) or from malicious scanning.

By using network telescopes, many large-scale internet threats can be detected at an early stage of their infection cycle. Using certain worm signature auto-generation tools (such as Autograph (Kim and Karp (2004)), PAYL (Wang et al. (2006)) or WormShield (Cai et al. (2007))), network telescopes can have a large impact in stopping the rapid spread of a worm.

Tarpits

Tarpits, as described by Williamson and Williamson (2003), Li et al. (2004),Weaver et al. (2004) and Yegneswaran et al. (2004). are another form of random scanning worm defence, in some ways similar to network telescopes (see above). LaBrea Tarpits (2008) is one of the most popular software implementations of this concept. A controller host on a network listens for ARP requests that go unanswered as no host on the network is associated with that IP - showing that the requester is either misconfigured or malicious, as above.

The tarpit controller then proceeds to send a SYN/ACK reply on behalf of the non-existent host, initiating a “three-way handshake”. Any further contact with the sending host is ignored, and no socket is opened - the malicious or incorrectly configured host will wait for the connection to timeout before continuing with any further connections on that thread. This timeout results in a large slowdown of major internet threats, giving more time for further countermeasures to be implemented.

Helpful Worms

A controversial form of “strikeback”, a helpful worm is a means of countering a malicious worm.

Malicious worms use security vulnerabilities to infect hosts on the Internet, and will often leave the vulnerability open for further infections if the host is disinfected but not immunised. In some cases, malicious worms will close the security hole and open another, in order to receive commands from a controller of some sort.

Helpful worms use these security vulnerabilities to infect vulnerable hosts, then close all vulnerabilities, before going through the same malicious worm cycle of infection. However, once they have propogated for a certain period of time, they delete themselves, leaving the host clean and protected from further worm infection.

Though the concept seems sound, in practice helpful worms can often result in equally massive traffic loads as malicious worms, and in some cases can do even more damage. The classic example of a helpful worm that resulted in rising infection problems is the Welchia worm, designed to counter the Blaster worm. It has been documented in Perriot (2008), where it is noted as having a “damage level” of “moderate”.

4.3 Early Simulations

The following two sections are a documentation of the process of early simulations that took place, and, in the case of Section 6.2, a comparison between simulation results and live data.

4.3.1 Initial Simulations and Component Construction

In order to thoroughly test the simulator, simulations of increasing complexity were run, beginning with very simple connectivity simulations and data transfer. Plugin development was an iterative process that began with very simple components. A single set of plugins, 'Default Action', 'Default Node', etc. was constructed. These plugins were initially very rigidly defined, with very little variability. After some time, a hashtable was added, by default, to all major components of the system, allowing for much more semantic content to be added, for instance, by adding a 'bandwidth' key/value pair to a connection, a single basic plugin could be adapted to represent a range of connection types, from high-traffic, low quality cable to high-quality ISDN.

4.3.2 Component Evolution

Each of the components of the simulation described (nodes, connection, action and packets) have undergone independent evolution. Nodes and connections, being the focus of the research performed, received the most attention: actions and packets less so. The following section documents the evolutionary process that the components went through.

Node Evolution

As increasing complexity of representation became a pressing need for the simulator, so the complexity of the node component scaled. The node components were the set of components which required the most attention, as they represent complex pieces of machinery.

Initially, all that a node represented was the end of a connection. The sole attribute of a node was the list of connections which terminated on that node, in order to 'fetch' packets off the connections.

As more semantic requirements were being made of simulations, so properties were added to the system.

The first major 'hard wired' attribute was an IPv4 address, so that nodes could semantically know their 'names', and address packets to other nodes based on a logical naming scheme.

Once logical connectivity had been achieved, practical operations had to be added to the nodes. Outgoing and incoming packet queue were added to represent packets arriving and leaving a network interface (though initially it was only a single property, and later developed into a full queue), and a set of methods were created for handling packets on these queues - initially, 'interpret packet' and 'send packet', while the packet queue only held one packet at a time, but later ‘queue packet’ and ‘accept packet’, which placed packets onto the queues.

After testing, it became evident that two distinct forms of node would be required in order to simulate as large a portion of the Internet as possible without requiring an unreasonable amount of resources. This led to a divergance of node types: a Host node represented a single host in the real world, while a Network node represented a network of one or more hosts, which interacted on an abstract level. Because of the abstract nature of a Network node, much internal processing could be reduced to simple mechanics: every time period in the simulation, the network nodes’ internal working were assumed to operate as if they were a connected series of Host nodes. This also led to an abstraction which could lead to a more convenient grid processing solution, by treating all nodes that have been distributed over a grid as a Network node, and incorporating all their input and output as the Network nodes’.

Connection Evolution

Connection objects were created to represent either logical or physical connections between hosts, depending on the necessity of the simulation. This definition, while very broad, leaves very little implicit actions and properties of a ‘connection’. As such, the core definition of a connection has not evolved past a send action, and two hosts that the connection links to.

In order to allow for latency and bandwidth, a queue was added to connections. This was later adjusted to be a vector, as realistic conditions for IP packet arrival could involve packet arrival in a different order to the order in which packets were sent.

Once the base connection object was completed, specific types of connections were implemented. Broadly, the two schools of connections were ‘immediate’ and ‘delayed’. Immediate connections, though less realistic, used less randomness in their sending procedure and so were more useful for testing. They also required less computation and memory, as packets would arrive immediately after sending, be interpreted, and then disposed of, compared to delayed connections where messages were stored for some period of time.

Action Evolution

Actions as a base object have remained largely unchanged since the original implementation. Initially, an action was a standard object (thus containing properties for name and version, as well as a tag hashtable for customisability), with a single run method that was executed when the event scheduler reached the action in the execution sequence. Eventually, this was amended to include a possible override for passing a message to the run method, allowing more specific commands to be executed on a robust action object.

The primary example of this was in message passing - if an action was created that passed a message between two nodes, it would (without some means of specification) be tied to only two nodes - a “send” action would be required for every connection between one node to another. By adding message-passing functionality, a generic “send” action could be created which could then use the message to interpret which node was sending and which node was receiving.

Packet Evolution

Packets are also largely unchanged from the original design decisions: a packet object contains a string to identify the receiving and sending node objects, and a string as a message.

The standard packet object used for simulations run to date is an IPv4 packet type - the recipient and sending node references are replaced by an IP address datatype, but no other functionality has been added.

4.4 Distributed Memory Prototype

GraphSim, based on discrete, divisible networks, is an ideal candidate for distribution over a logical structure such as a computational grid. This would this make the simulator capable of running on systems superior to a desktop or even a single high capacity server, thereby allowing levels of detail in simulations that could not be practically implemented on systems without as much power.

However GraphSim was designed with single desktop computer use as a priority. Making the system grid-distributable is not necessarily contrary to this priority, but it would require a reconstruction of the underlying memory allocation system on which the simulator runs.

Throughout the length of this research, memory-use has remained the bottleneck in simulations. When the number of packets and simulated hosts in a system that models malware grows exponentially, the memory capacity of computers used rapidly becomes insufficient. Simulations performed in Chapter 5 became untenable (taking over 18 hours per simulated tick) once more than one million packets were being simulated. While many computers were available for this purpose, few had more than four gigabytes of memory. While more than adequate for most simulations, Internet worm modeling (discussed more in Chapter 4) requires very large quantities of memory, as worm scanning will refer to (and thus instantiate or at least require record for) many nodes and grows exponentially with the worm infection rate. Section 3.6.2 considers the challenge of memory shortage and points out that a full IPv4-space simulator would require at least 128 gigabytes of memory. Assuming that computers available for simulation had 2 gigabytes of memory, this would mean that 64 computers would be required.

As a result, it was determined that grid distribution of memory was an important concept for prototyping.

The original grid prototyping ran the entire simulator on every grid node that was to be used. Lock-step processing (which would have slowed the simulation down due to network input/output latency overhead) was avoided by executing a different simulation on the "slave" grid-computers (while a controller distributed simulation actions to perform), which run an infinite loop and instantiated nodes in memory when network packets arrived specifying the address of the node. Furthermore, the representational challenge of this prototype added an additional layer of complexity. By distributing each simulator to a separate computer and allowing asynchronous simulation, the inconsistency due to minor heterogeneity of simulators became a concern. Because the simulators were operating on slightly different systems (even if the hardware and software were originally in a consistent state, networking and other physical aspects of the systems would result in a non-uniform execution of the simulation), the results would differ from those which would be generated from a single large system, unless the simulators would operate in lock-step. .

Because processing was not the bottleneck (commented on in Section 3.6.1), latency was not considered to be a problem. Instead, the memory overhead of maintaining multiple running execution engines became the primary concern.

The final grid-prototype that was used acted as a "node server", which operated as an entirely seperate program. The entirety of its function was to store address and node information and return it, and no simulation was performed at any stage. By distributing the storage of nodes and packets addressed to those nodes to the node servers, the core simulator could be used for execution while holding only those nodes which were to be simulated in detail. By using a modified network node (discussed as an evolved form of node in Section 4.3.2), which accepted packets to a network node and then sent them as data to the storage nodes, it was possible to use an arbitrary number of node servers. Intuitively, it was easiest to divide address space into equal sizes, so the preferred number of node servers would be a power of two. The prototype model used two node servers during development, and tested the extensibility using four servers.

Figure 4.1 explains this concept: the original memory model used a single computer's memory for storing node components. By distributing the memory use, multi-

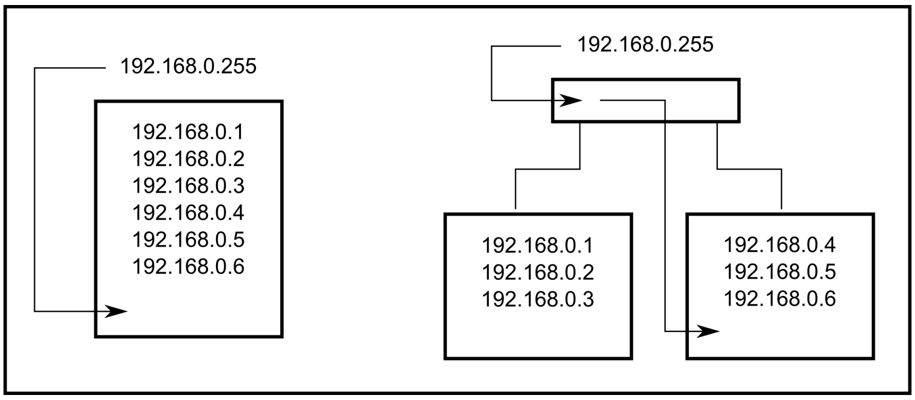

Figure 4.1: Distributed Memory Allocation System

ple computers can share their memory for node storage. The right half of the diagram demonstrates that certain boundaries exist which determine which computer receives specifically addressed nodes - in this case, the left "node server" receives nodes with addresses lower than 192.168.0.4, while the right server receives those of that value or above.

Figure 4.1 also demonstrates the abstraction in the memory access - from the programmatic perspective of the simulation, it has stored the node in local memory. By abstracting memory access via nodes that represent entire networks, simulations can "transparently" access more memory than their local systems have available using a distributed system.

It was found to successfully improve the memory allocation significantly, but as a non-core component of this research, was not investigated any further.

4.5 Summary

In this chapter, the conceptual design of a robust network simulator has been drawn. The choice of malware as the subject of research was covered in Section 4.2, with complementary discussion of countermeasure development in Section 4.2.4, focusing on a variety of methods available for research.

Once Internet worms were established as the focus of the research, the design of the simulator itself began in Section 6.5. Having constructed the prototype stages of the simulator, the evolution of the various modular plugins that were used was documented, with notably more emphasis on nodes and connections than on packets and actions. This work is evaluated in Chapters 5 and 6 which show the functional aspects of the simulator, and its applicability to solving real-world large scale issues respectively.

5 Testing, Calibration and Network Simulations

5.1 Introduction

This chapter will document the planning, development and results of the earlier simulations that are used by the developed simulator.

Section 5.2 provides a broad overview of the specifics used in networking for this simulation. Section 5.2.1 deals with the specific challenges of IP addressing, while Sections 5.2.2 discusses the parameters of specific elements of the simulations. The remaining documented parameters give specifications for the engine to repeat the simulations described later in the chapter.

Section 5.3 explains the format that this research uses for documenting simulations, and Section 5.4 documents the simulations themselves.

Finally, Section 5.5 concludes the chapter with an overview of the executed simulations.

5.2 Network Parameters

If simulations for research are to be run on the simulator framework, a careful definition of the parameters and definitions of the simulation's environment needs to be documented, in order to ensure that these simulations can be repeated for testing purposes.

This section will cover the overarching parameters of the system, fundamental to any basic simulation. These definitions are necessary before discussing the simple simulations covered in Section 5.4.

5.2.1 IPv4 Addressing Parameters

IPv4 simulations present many challenges, perhaps the most significant of which is in the structure of networks. In Paxson and Floyd (1997), the challenge of simulating heterogeneous networks is discussed.

It was decided that a simple tree structure for the network, using basic subnetting, would suffice for network structures. The Internet would be divided into 256 class A networks, which would be divided into a further 256 class B networks, and so on. Each subnet would have a 'gateway' host, the first addressed host on the network (thus the gateway for 192.168.0.0/16 class C network would be 192.168.0.0, and the gateway for the 146.231.0.0/16 class B would be 146.231.0.0). This also means that the 'parent' node, through which all of the highest levels of packets pass, is 0.0.0.0. This presents a significant bottleneck if the number of packets forwarded in a discrete time unit is limited to a certain quantity.

This is obviously a vast oversimplification of real world network structures, however for simple simulations, it can be considered sufficient.

Nodes in the IPv4 network are all implemented as subclasses of the abstract 'IPv4' node class, thus controlling nodes that may be added into such a network. By definition, they all contain an address, as well as a connection link to their 'gateway'.

In order to simulate the way in which a default route works, all packets are addressed to the intended recipient, then sent via an IPv4 connection object to the 'gateway' node for the network on which the sending node is located. On message interpretation, the gateway node then decides what to do with the packet, depending on the packet's recipient node:

- If the recipient node is directly connected to the gateway that receives the packet, such as 192.168.0.0 receiving a packet for 192.168.0.15 (in which case the message is forwarded to the recipient)

- If the recipient node belongs to a different subnet, such as 192.168.0.0 receiving a packet for 146.231.1.15 (in which case the message is forwarded onto the gateway's own 'default route' to be processed again)

- If the recipient node is within the gateways subnet but not attached to the gateway, such as 192.0.0.0 receiving a packet for 192.168.0.1 (in which case the message is forwarded to the next logical gateway in the sequence)

- If the recipient node is the gateway, then it interprets the packet as a standard node

In order to reduce memory consumption, network nodes have also been developed. Network nodes, another IPv4 sub-class, allow IPv4 packets to be received, and represent any form of network, at any point in the IPv4 structure. Because of its abstract nature, a network node could represent forms of network other than the tree type that the one-to-one simulated hosts represent.

5.2.2 Host Parameters

When considering hosts upon the simulated Internet, certain assumptions must be made about the the hosts' properties. Because many of the worms that were to be simulated required either a certain operating system or a certain software package to be installed, when node objects were instantiated they had random properties associated with them.

Two sets of assumptions have been made when determining the properties of nodes:

- The first set of assumptions regards the actual distribution of properties. No accurate data is obtainable about the nature of computers connected to the Internet - in Paxson and Floyd (1997), the authors comment on the largely unmeasurable nature of the Internet. As a result, the values used were chosen with as much accuracy to the current state of the Internet as was available.
  The values were chosen with influences from Pang et al. (2004) and Net Applications (2008), researchers who have studied the Internet and attempted to make inferences about its composition.

- The second set of assumptions is speculative - it is assumed that at some point further development may take place using parameters that have not been used in the simulations discussed in this document. As a result, parameters specified here may include operating systems, software, or other information not immediately useful to this research, but may be useful for later simulations.

When a host is instantiated, it begins with randomly determined properties. Some of these properties that are stated are not used in the simulations in this research - this is for easier later development, and the chosen properties can easily be edited, changed, removed or added to by adjusting values in a configuration file. These properties are as shown below:

- 70% chance to be a Windows OS
  If so, then:
  - 5% chance to be Windows Vista
  - 40% chance to be Windows XP
  - 35% chance to be Windows 2000
  - 20% chance to be some other version of Windows
  - Whichever version of Windows, there is also the following chance of software being installed on the system:
    * 10% WebDAV (for Welchia infection)
    * 30% IIS version 4 (for Code Red vII)

    * 5% ISS (for Witty)

- 20% chance to be a GNU/Linux OS
  If so, then:
  - 60% chance to be Ubuntu Linux
  - 20% chance to be Fedora Linux
  - 20% chance to be an unknown Linux distribution
- 10% chance to be some other OS
  If so, then:
  - 50% chance to be some version of FreeBSD
  - 50% chance to be some other, less popular OS

5.2.3 Protocol Parameters

TCP/IP is the only protocol on the OSI stack to be simulated, corresponding to layer three and four. This corresponds to the form of worms simulated - they operated primarily using TCP/IP.

Only IPv4 was simulated - IPv6 would be a simple extension, but was not included due to intial concerns regarding the large scale. It could be explored further using a large enough system of the Grid based simulator as discussed in Section 4.4.

5.2.4 Traffic Parameters

Non-relevant network traffic was not representationally simulated. Latency and bandwidth limitations are implied as delays and queues are in place, but due to efficiency challenges, they were not included in the simulations.

5.2.5 Time Parameters

Simulations were typically run in time 'ticks', running from time one through ten million), though in several cases where execution time became unreasonably high, simulation execution times were reduced to one million ticks and these cases are noted in the specific experiments.

The time it takes for a packet to be sent over a single network connection is a single tick. This can be increased (and is for some of the later malware simulations) in order to model latency, showing that the system is configurable. Because time is abstracted in these simulations, the meaning of a tick depends on the simulation - future users of the system can associate a tick to suit their simulations' requirements.

5.3 Test Format

Tests that will be performed will be detailed using the format below.

5.3.1 Statement of Subject

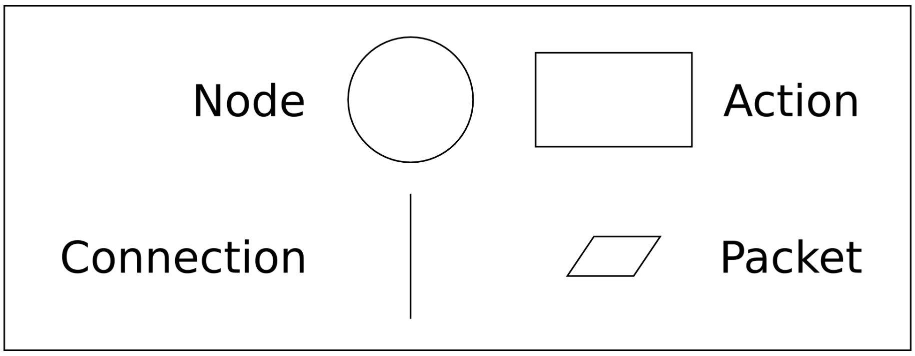

Figure 5.1: Simulation 1: Creating Components

The subject of the simulation is clearly stated in this section, specifically for purposes of later testing. By stating the subject before simulation takes place, the results may refer back specifically to the subject statement and thus be evaluated objectively and independent of any other notable results observed in the simulation.

Where appropriate, diagrams will be included in the subject statement to clarify the setup of the simulation. The symbols used in the diagrams can be seen in Figure 5.1.

5.3.2 Statement of Parameters

Any parameters specific to a simulation are outlined so that later researchers may repeat these simulations for their own validation. This section will be as thorough as possible, but will focus specifically on parameters that are different from those used previously, or different from those normally used, detailed in the previous chapter.

5.3.3 Statement of Results

The results of the simulation will be stated, and shown in summary (either in table form, in statistical analysis, or by visualisation). At this stage, conclusions will not be drawn from the results further than noting anomalous behaviour or marked differences from the expected results.

Where appropriate, ellipses have been used to reduce sample outputs for the sake of brevity. In these cases, additional result examples may be found in Appendix B.

5.4 Component and Network Simulation Testing

Before the simulator can be used to test advanced concepts (such as Internet worms), it must first be confirmed that it's basic components and fundamental operations behave in an expected manner. The following tests were devised to determine this.

5.4.1 Rationale for Simulations

The simulations in this section were selected as a series of increasing complexity, testing the fundamental components and behaviours of the simulator. The first simulation, component creation, tests that the simulator is capable of creating and maintaining components in memory. The second simulation, connectivity testing, tests the ability of the simulator to successfully communicate over the network. The third simulation, routing, extends the communication testing to a non-trivial level. Layered networking tests the ability of the simulator to model modern multi-protocol networking by modeling protocol encapsulation, which is an imperative part of a simulator that can scale resolution. Network node simulation also tests resolution scaling by abstracting an entire network, an important aspect in Internet-scale simulation. Finally, the last simulation of this chapter combines routing and network nodes in preparation for the testing in the following chapter which will make use of this pattern.

5.4.2 Component Creation

The first simulation that must take place is the creation of the components used in the system. Nodes, connections, packets and actions must be instantiated, and tested thoroughly within the bounds of the simulation. The core functionality of each (nodes' ability to hold information, connections' capabilities to connect nodes and so on) must be tested as well.

Statement of Subject

In this test, the creation of components is the core subject. Each type component should be created and added into the system, then tested to see whether the simulation can maintain these components intact in memory for the duration of its execution.

Furthermore, the claim that arbitrary parameters can be added to components should be tested - some components must have semantics added and later extracted to determine whether they remain stable in memory.

Statement of Parameters

The test shall create a node, and assign it an address (an arbitrary piece of information associated with the node, as not all nodes have an address intrinsically). It will then attempt to extract this information from the node and test whether it remains unchanged since input.

The test must then create another node, and create a connection that connects it to the node that has already been created. A packet component should then be created, and a message associated with it. Finally, an action component should be instantiated and added to the simulation global scheduler - it should test whether the address associated with the nodes remains the same after some simulated period of time has passed.

Statement of Results

The output from the simulator is as shown below:

```
Node created
Address assigned
Address remains in the node
Created and connected another node
Creating default packet
Created a packet
Created an action
(A time tick passes)
Address A remains the same over time
Address B remains the same over time
```

This output (which uses a more verbose set of components than those used in later simulations) explicitly shows each stage of the component instantiation and testing process, demonstrating successfully that components of all types can be created in the system, and their operation remains stable over time.

5.4.3 Connectivity

Connectivity needs to be tested as a core part of the networking simulation. Connection objects would have been created as part of the last simulation test, but connectivity must test the ability of the network to transfer packets between hosts. This level of simulation does not require any specific actions to take place when packets arrive, or generate more than a few packets - it just has to send them correctly from one node to another, via a connection object.

Statement of Subject

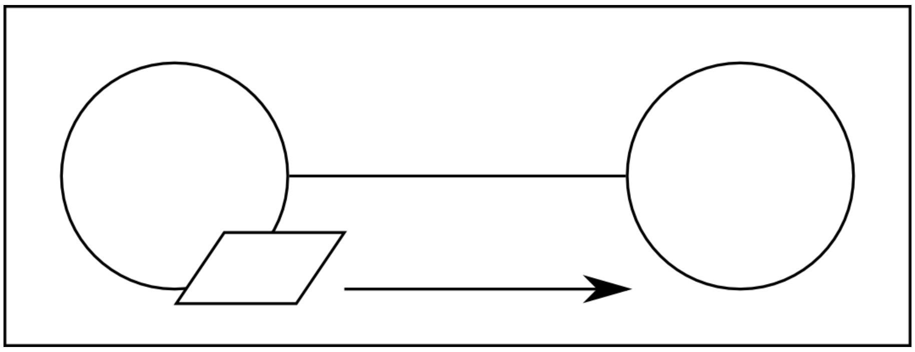

Figure 5.2: Simulating Connectivity

The focus of this test is to determine whether connectivity - a core concept in networking - operates correctly and successfully in the simulation. The simulator must create nodes and connections, then pass packet components over those connections. The node behaviour must be set to test and report the payload of any packets it receives in order to determine whether the payload remains intact.

Statement of Parameters

The simulator must create some node objects (in the simplest case, two), and a connection object, connecting the nodes. It must then create a packet object with a “payload” or messsage, and queue its sending on one of the nodes. The connection object must then be used to extract the packet from the node, and successfully transport it to the other node. Finally, the receiving node must extract the packet from its incoming queue, and interpret it by testing whether the payload retains integrity.

Statement of Results

The output from the simulator is as shown below:

```
Sending packet.
Packet arrived, payload: Packet Payload
```

This output shows the sending node stating that the packet is in the process of sending. When the receiving node receiveds the packet successfully, it prints the payload - this shows the simulation successfully sent the packet.

5.4.4 Routing

Once connectivity has been established, routing can be implemented. As part of the implementation of routing, it is required that several important aspects of modern networking are also included, particularly addressing and the concept of a “default route”.

Statement of Subject

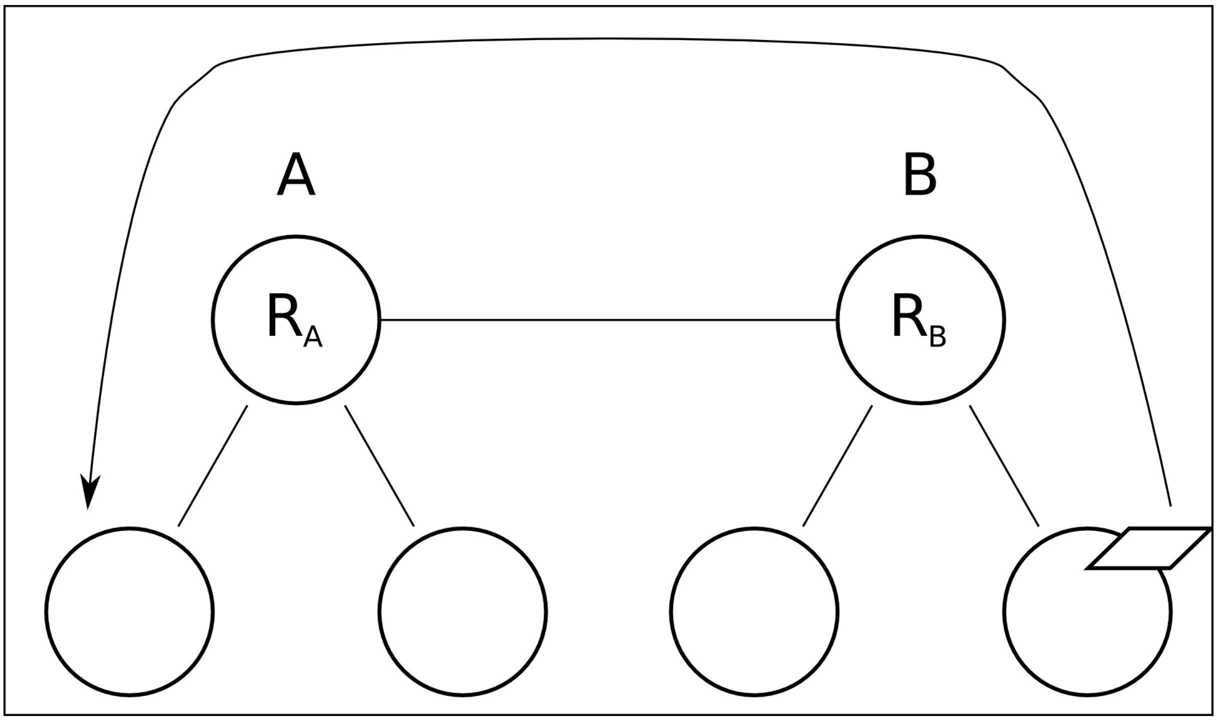

Figure 5.3: Simulating Routing

The test requires multiple nodes created, explicitly *without* a mesh network, as shown in Figure 5.3. Through the use of default routes and routing tables, it is possible for messages sent from host A to be sent to host B when A and B are not directly connected, but a route does exist that connects them both.

The concept used for routing, in this case, is by attaching the attribute “default route” to nodes in the simulation, stipulating which host should receive packets should they not be directly connected to the host that the message is to be sent to. Hosts in the system also require a “routing table” which specifies which hosts they are connected to, or which hosts in the route will eventually lead to the destination. Once a packet arrives at a routing host, it can be forwarded to the correct remote router, which can then send the packet on to the specific destination host.

Statement of Parameters

For this simulation, nodes can have one of two behaviours, depending on which attributes have been assigned to them: if a node does not have a routing table, it will pass all packets it receives to the node specified in their “default route” attribute (which all nodes have), but if it has a “routing table” attribute (which stores specific nodes with their addresses as keys) and the destination is in the routing table, it will attempt to send the packet to its specified destination.

When the simulation starts, it creates two networks (called A and B for this explanation), each containing two “normal” nodes (with no routing tables) and one “router” node (which has routing tables). It connects the normal nodes of A to the router node of A, and similarly connects the normal nodes of B to the router node of B, then connects the router nodes - all via connection objects. Finally, it creates two packets, one in network A, another in network B, addressed to nodes in the network B and A respectively. They are then instructed to send these packets.

The nodes in network A and B are set with their routers as default routes, while the routers have their default routes set to the other router (which is unrealistically simple, but will suffice for testing).

The packets should arrive in their respective routers, then, when testing shows that the destinations are not found in the routing tables, should be forwarded to those router’s default routes. When they arrive in the remote routers, they should then be forwarded to the destinations specified in the routing tables, and so then finally arrive at their destinations.

Statement of Results

The output from the simulator is as shown below:

```
Packet arrived. Routing. (Packet 1 Router A)
Packet arrived. Routing. (Packet 1 Router B)
Packet arrived. Payload: Packet Payload 1
Packet arrived. Routing. (Packet 2 Router B)
Packet arrived. Routing. (Packet 2 Router A)
Packet arrived. Payload: Packet Payload 2
```

This output shows packets arriving at various nodes. Where routing takes place, the router is stated in parantheses at the end of the line. If the packet arrives at it’s correct destination, the payload is shown.

5.4.5 Layered Networking

Modern networking uses a stack abstraction to model the various layers of protocols used in communication, detailed in Chapter 2. The conceptual model of a networking “stack” is implemented by wrapping high-level protocols’ traffic in low-level protocols packets. The simulation of this concept can be performed by allowing the “message” of the packet objects to be a generic datatype, which can (via object-oriented inheritance of packet objects the generic datatype) include higher-level packet objects.

This simulation test will test the simulator's capacity to include traffic at any level, while encapsulating the data from higher levels.

Statement of Subject

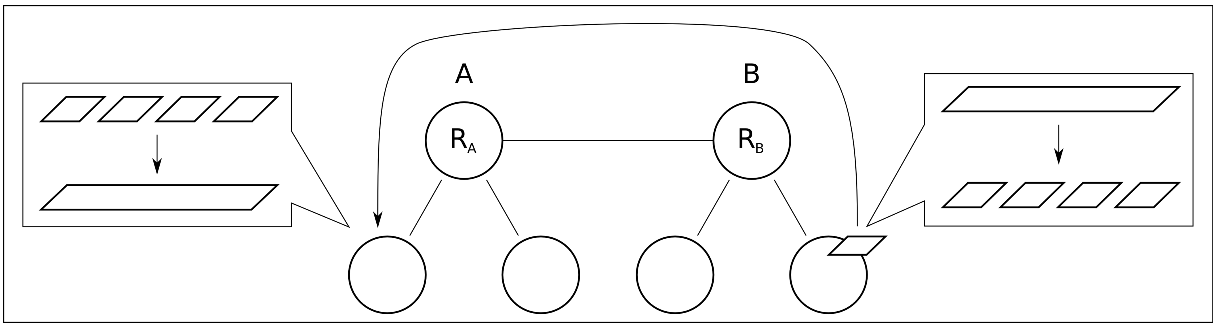

Figure 5.4: Multi-Layered Network Simulation

In this test, the focus is on representing real-world multiple-layered protocols. This is represented by introducing several levels of detail specific to certain layers of networking.

Packets no longer include addressing as a seperate sub-entity: addressing is now made as part of the message, and extracted as a header. This is also complicated by adding an optional (dependant on size) fragmentation element to the way in which messages are passed. Messages, in this test, must be able to be passed using a high-level of abstraction (to represent a high-level protocol using networking libraries to abstract networking details), such as "Send the message 'example' from host A to host B", but be 'wrapped' in a lower protocol in the networking stack (as they would be by those libraries) and be reassembled into its initial state on final reception. This process is demonstrated in Figure 5.4.

This simulation passes a message with a high-level abstraction, which is then wrapped in an addressed header and fragmented if necessary (as it would be using TCP/IP). On reception, a node will "interpret" these packets by stripping the headers (but retaining the information they store), reassembling the data, and finally displaying the results.

Statement of Parameters

The simulation will use the network assembled for the previous test: four "normal" nodes connected to two routers that are themselves connected. Two messages will be prepared: one that is larger than 1500 bytes (which ensures that normal IP will fragment it into multiple packets), and one that is smaller. Both must be sent, but must be transformed into the multi-layered form. Once they arrive, both packets must then be examined and display the original message used.

Statement of Results

The simulator successfully routed packets between two non-adjacent nodes on the network (using the routing established from the previous simulation). By displaying incoming packets at a "raw" level, then once they had been stripped of headers, then again once a complete message had been reassembled, it was clear that multi-protocol simulations were possible.

5.4.6 Network Nodes

One concern of simulation is the efficiency of memory and processor use. One of the most common ways to improve on this efficiency is abstraction of large portions of the network. Network nodes are nodes that play the role of abstracted networks. By allowing a single node to behave as a whole network, the focus of the optimisation of the simulator can move from node size and access to packet movement.

The abstraction of network nodes, however, can only hope to retain accuracy if they can operate similarly to networks (in terms of the parameters of the simulation). This added challenge must therefore include elements of randomness (with associated probabilities to ensure measurable similarity to the object they model) that will lead to possible differences from observed data.

Statement of Subject

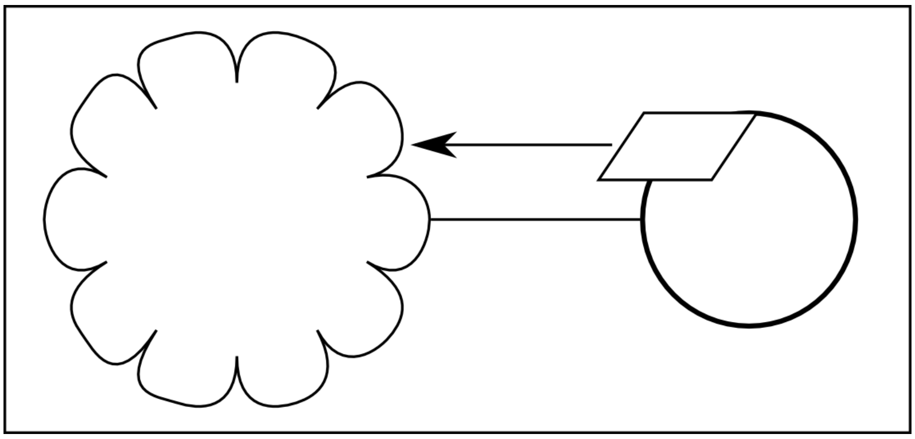

Figure 5.5: Simulating a Network Node

In this simulation, a network node must be created and attached (via a standard connection object) to a normal ("host") node (as shown in Figure 5.5). It should receive addressed packets and interpret them by storing the information efficiently (in this case, by storing the packet information but discarding the packet payload). Once the simulation has completed, the information stored by the network node must then be displayed for analysis.

Statement of Parameters

A single host node and a single network node must be created, and attached via a simple connection object. The network node's behaviour can be defined as pure storage at this stage: it merely records the destination addresses of packets that it receives. For testing, several million packets should be sent to show that a network node is an effective means of packet information storage.

Statement of Results

```
[...]
15.63.177.88 -> 2
194.6.13.195 -> 2
26.76.165.177 -> 2
59.1.6.220 -> 2
Single IP hits -> 1999082
```

This output shows the number of received packets at each IP address (sorted in descending order), and accumulates the number of IPs that received only a single packet. This shows that a wide variety of broadly-spaced IPs received packets.

5.4.7 Mixing Node Types

The example shown in Simulation 5 (Section 5.4.6 above) initially appears contrived - routing has been removed from the example, so all packets sent to the network node are logged - even packets which are not destined for the network that it represents.

To display network nodes in the context of a complete network simulator, a more elaborate example has been constructed using several of the networking simulation fundamentals already tested. A network with several nodes, each with their own routing and seperate node information should be constructed for this test, and a large number of packets originating from each of them with a variety of destinations (both for the network node and for the other hosts) allows a more stringent framework for testing the usefulness of these components.

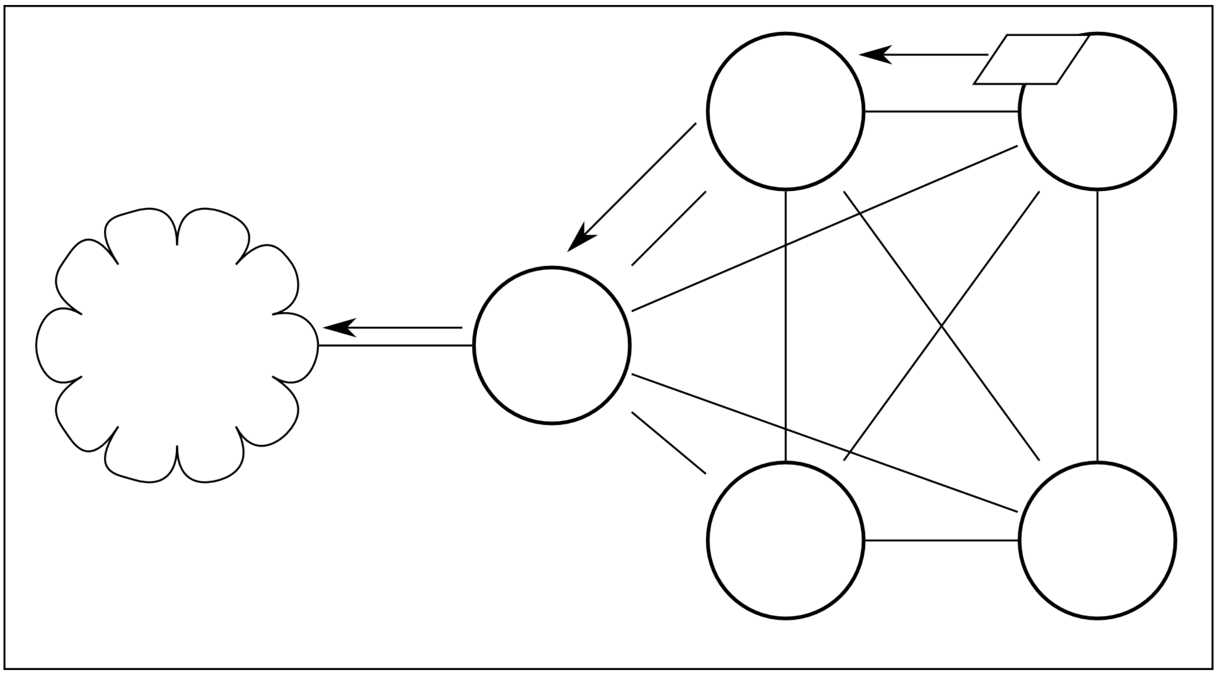

Figure 5.6: Simulating a Mixture of Components

Statement of Subject

This test exists to bring together the individual aspects of networking simulation that have been used so far and to construct a simulated network that can be used for later simulations. Several host nodes (in the example used here, sixteen) should be created and semantic information should be inserted, such as addressing, routing, etc. They should be mesh-connected, and one of them should act as a router with an "Internet connection" as a default route to a network node. This network node, then, should collect packets and store their information.

A large number of packets should then be created (several million) and placed in the outgoing-queues of randomly selected host nodes on the network. These packets should be randomly addressed (some specifically to other nodes in the local network, others specifically outside of the network), and once placed in the queues, should immediately be sent.

Once the simulation has completed, the network node and the host nodes should report on the number of packets they have received.

Statement of Parameters

The above statement of subject covers all required details and parameters for the purposes of this simulation.

Statement of Results

The results have been reduced for the sake of brevity, see Appendix B for additional results.

```
[...]
79.71.252.97 -> 2
51.98.142.195 -> 2
87.204.163.26 -> 2
57.98.78.255 -> 2
17.197.37.120 -> 2
Single IP hits -> 979733
```

This output shows the number of received packets at each IP address (sorted in descending order), and accumulates the number of IPs that received only a single packet.

5.5 Discussion of Results

In this chapter, aselection of tests were applied to the simulator developed for this work in order to test the basic functionality. Simulations began with simple system tests to confirm the basic operations - plugins and communication - and then advanced to model routing and protocol stacks, finally modeling a single complex network connected to a simulated Internet. The simulations found that the simulator was working as expected.

5.5.1 Simulation Results

The results of the networking tests are all as expected - the networking functions of the simulator work as designed, and are capable of handling large volumes of traffic (millions of ticks and packets).

5.5.2 Simulator Results

The results discussed in this chapter show that the simulator meets the goals set out for it in that:

- it is capable of a selection of network simulations
- it is able to operate at several levels of abstraction,
- it is scalable from the level of directly simulated networks and with the use of abstraction to the operation of networks simulations at the Internet scale.

The simulator that was developed also met several less tangible goals which were set out for development (see Section 2.6.2): It was shown to be highly modular (with many plugins already written for easy use by potential researchers - See Appendix C for a brief listing), and exceptionally platform independant, executing simulations on at least three different primary operating system platforms. Actual test implementations were run on Windows XP, Windows Server 2003, FreeBSD and Ubuntu Linux.

Finally, it was found that the simulator also supports rapid plugin development. Node plugins, requiring the most complex development, typically required rewriting up to two methods, each of approximately one hundred lines of C# code. Connection and action plugins required less development, and packet objects never required any. This is encouraging and meets one of the most important high-level goals of the work - to be useful for rapid response security simulation in research.

5.6 Summary

The research and simulations performed in this chapter show that the basic assumptions that the simulator needed to fulfill in order to begin development of complex simulations were in place. The simulator in Section 5.4.2 was shown to be capable of instantiating a variety of modular plugins successfully. Section 5.4.3 demonstrated that it could successfully model connectivity between two nodes, and Section 5.4.4 extended this with packet routing. Section 5.4.5 demonstrated that the simulator was capable of layered networking by wrapping data of a different protocol within a packet. Finally, Section 5.4.6 brought all these concepts together into a complex simulation. All of these simulations made use of the parameters stated earlier in the chapter, in Section 5.2.

Based on the validation of the functional components of the simulator, the following chapter presents simulations of malware and security countermeasures to that malware that test the ability of the simulator to model these concepts in real life.

6 Malware and Advanced Simulations

6.1 Introduction

This chapter continues with the simulations described in the previous chapter. Whereas Chapter 5 established the capability of the simulator to adequately represent networking at a varying levels of complexity, this chapter elaborates, with simulations meant to test the system's capabilities as a representational simulator.

Section 6.2 proposes the malware, worm and advanced simulations which will be used in this chapter.

Section 6.3 introduces additional parameters used for the malware simulation in this chapter.

Section 6.5 begins the documentation of simulations by introducing the visualisation technique that is used throughout this chapter.

Section 6.6 focuses on worm simulation, covering the simple simulated case then carrying on to simulate the Witty, Code Red vII, Blaster and Welchia worms, as discussed in Section 2.4.

As this is the core of the chapter, the contents are explored here in more depth:

Initially, a conceptual worm was simulated to explore and test the concept of an Internet worm. This is detailed in Section 6.6.1. This was then extended to model the relatively simple "Witty" worm, in Section 6.6.2. The Code Red vII worm added degrees of complexity in scanning algorithms, and its simulation is described in 6.6.3. Finally, the Blaster and Welchia worms are detailed in Sections 6.6.4 and 6.6.5 respectively.

Section 6.7 introduces two "advanced" simulations - simulating a malicious worm and a helpful worm simultaneously to gauge the effectiveness of this counter-worm strategy, and modeling the effects of a network telescope.

6.2 Proposed Worm Simulations and Expected Outcomes

As a standard set of plugins was being constructed, a toolkit for worm simulation became a possibility. A 'worm propogation' action component was constructed, and was designed for modularity. Once development of a general worm concept had been completed, the worms to be modeled were selected. Presented here are the worms that were chosen for simulation.

Simple Worm Simulation Showing the effects of a simple worm that randomly scans and infects (such as that documented in Section 6.4.1), this initial simulation case was designed as a test for the use of worms in the simulation, as well as a starting point for further worm development. Simple scanning and perfect infection rates allow this simulation to show the full effects of a random scanning worm, and testing the full effects of a worm that could infect a large portion of the Internet very quickly.

Witty Simulation Described in Section 6.4.2, the Witty worm should show very rapid propogation compared to other worms (due to its connectionless infection), but the spread should be more random. This worm adds an additional level of complexity to the simulation by adding non-perfect infection. Documentation on this worm can be found in Section 2.5.3.

Blaster Simulation The first 'real worm' simulation to be run, simulating the Blaster worm that is described in Section 6.4.4. This simulation adds additional layers of complexity to the simulator by using a more complex scanning algorithm, and adding additional requirements for the infection determination. In the case of a single host simulated, it is expected to see a large quantity of packets sent to a single class C netblock. For more accurate simulations, many full class C networks should be completely scanned. Documentation on this worm can be found in Section 2.5.3.

Code Red II Simulation Described in Section 6.4.3, the Code Red vII worm should display more traffic in the local networks than the Blaster simulation mentioned above, for each single host infection. This worm adds additional complexity to these simulations by adding a more complex scanning algorithm, resulting in observable patterns in the results which aid in testing. When simulating the effects of Code Red vII over a fully simulated Internet, class A and B networks should show much higher saturation than for Blaster or Simple worm cases. Documentation on this worm can be found in Section 2.5.2.

Generic Red vs. Blue Simulation In testing malware countermeasures, Red vs. Blue worm simulations are of interest. By simulating two worms at once, this simulation stresses the simulator and tests it's capability to simulate more complex actions. Discussed in Section 4.2.4, this initial simulation will use Simple worms, as discussed earlier, for both the malicious (classed as "Red") and helpful (classed as a "Blue") worms. This simple case will test the mechanics of Red vs. Blue worms instead of trying to extract any interesting semantic information about helpful worms as a malware countermeasure.

Blaster vs. Welchia Simulation The Welchia worm (described in Sections 2.5.3 and 6.4.5) removed the Blaster worm from infected computers, being perhaps the most famous of the "helpful" worms (though it did much damage in the process, so the term "helpful" might be incorrect). This simulation will test the effects of these two worms upon the Internet. It is expected that in cases where the worm spread is at all limited, the intersection of the two worms' scans will be small, and show little effect. If the two worms are allowed free reign over full-scale Internet simulations, the amount of intersection should be much larger.

Blaster vs. Network Telescope Simulation Network telescopes (introduced in Section 4.2.4) are one tool that is used in the defence against network-aware malware. Network telescopes can be considered to be a special case of a Network node object, so addressed packets to the node may be handled in a different manner if the node is flagged as a telescope. In this case, certain host nodes upon the simulated Internet will have an additional parameter to those mentioned in Section 5.2.2 - it will be marked as listening to the telescope. If the telescope is scanned by an infected host, then the host's IP address will be added to a blacklist. Telescope-flagged hosts will ignore all packets received from blacklisted hosts.

This simulation demonstrates the simulator's capability to model non-worm complex components, as well as demonstrating a complex node type that has useful and non-standard behaviours.

The expected result of this is largely dependent on the worms running in the simulation. The Blaster worm, used in this case, is anticipated to have a slightly slower propogation rate.

6.3 Further Parameter Specifications

The simulations presented here require further parameter specifications to remain reproducable. In this section, additional parameters will be introduced.

The parameters from the previous chapter detailed in Section 5.2 remain valid throughout the simulations in this chapter.

6.3.1 Common Worm Parameters

A host, once infected, will continue to send out new packets continually (every time tick in which the worm “activates”, dependant on the forms of the worms) and eternally (never stopping for the entire simulation).

6.3.2 Setup Parameters

In the case of individual node simulations, the initial infection took place on a host with IP address 192.168.0.1, a host on the list of IANA reserved private network ranges (documented in (IANA)) - though technically it is simulated as directly “internet-facing”.

In the case of subnet simulations, the Class C network 192.168.0.0/28 is simulated, a subnet of an IANA-reserved network (explained in Rekhter et al. (1996)).

In the case of Red vs. Blue worms, the malicious worm’s initial infection will be on host 192.168.0.1, and the ‘beneficial’ worm initial infection will be on host 192.168.0.8.

These initial IP addresses are significant, as the visualisations of the scanning will show: many worms use localised scanning, so the simulated reserved network will receive the majority of the scan attempts.

6.4 Programmed Worm Parameters

6.4.1 Simple Worm

The simplest worm has perfect infectability, infecting hosts regardless of details such as operating system or software running. It may use either a random scanning algorithm (selecting the next host to infect independant of the previously selected host), or an incremental scanning algorithm (considering IP addresses to be 32-bit integers, starting counting from 0 up to $2^{32} - 1$). Results are discussed in 6.6.1.

6.4.2 Witty

The Witty worm was discovered on the 19th of March 2004 and used an exploit in ISS, an Internet security package, documented by Gatti et al. (2004), to gain access to a host.Once the host was infected, Witty would overwrite a random section of the hard drive, then send 20,000 UDP infection packets to randomly generated IP addresses. Because the infection was contained in a single UDP packet, it spread very quickly.

Once the 20,000 packets had been sent, it would seek to another random section of the hard disk and overwrite it, beginning the cycle of overwrite-infect again.

Programming Specifics

For the Witty worm, if a worm infection packet arrived at a node, it would only infect the host if its operating system was Windows (which we simulate as having a 70% chance of occurring), running ISS (which is simulated to occur on 5% of Windows hosts).

Using a Bayesian decision tree, it was then determined that the probability of a computer being infectable by the Witty worm is:

$$0.7 \times 0.05 = 0.035$$

So when an infection packet was received by the Internet node, it would be considered a successful infection of the Witty worm with probability 0.035. The simulation of the Witty worm is detailed in Section 6.6.2.

6.4.3 Code Red vII

The Code Red vII worm has been extensively documented, as shown in Section 2.5.2. Released on August 4, 2001, it exploited a buffer-overflow vulnerability in Microsoft IIS servers Microsoft (2003c). Its scanning algorithm, explained in Friedl (2001), selects new hosts to infect by doing the following:

For the purposes of this example, consider a Code Red vII-infected host with IP address $A.B.C.D$:

The infected host generates a random IP address of the form $W.X.Y.Z$. It then proceeds randomly, by selecting a value between 1 and 8, and scans as follows:

- If the value is one, it will scan $W.X.Y.Z$
- If the value is between two and five (inclusive), it will scan $A.X.Y.Z$
- If the value is between six and eight (inclusive), it will scan $A.B.Y.Z$

If the randomly generated address falls into the 224.0.0.0/8, or multicast address space, or the 127.0.0.0/8, or loopback address space, then it will reselect the random address.

Programming Specifics

For the Code Red vII worm, if a worm infection packet arrived at a node, it would only infect the host if its operating system was Windows (which had a 70% chance of occurring), and it had IIS installed (which had a 30% chance of occurring).

Using a Bayesian decision tree, it was then determined that the probability of a Windows host running IIS was:

$$0.7 \times 0.3 = 0.21$$

So when an infection packet was received by the Internet node, it would be a considered a successful infection of the Code Red vII worm with probability 0.21. The simulation of the Code Red vII worm is detailed in Section 6.6.3.

6.4.4 Blaster

The Blaster worm was discovered on the Internet on August 11, 2003, and uses an exploit in Microsoft Windows' DCOM RPC Interface Buffer to gain access to Windows XP and Windows 2000 computers, as explained by Dougherty et al. (2003) and detailed in Microsoft (2003a) and Microsoft (2003b).

The Blaster worm forms the scanning target's IP address through one of two ways, according to Knowles and Perriott (2003):

- There is a 40% probability that it will choose an address $A.B.R_1.R_2$, where A and B are the first and second bytes (respectively) of the address of the host that is sending the attack, and R_1 and $R_2$are randomly generated in the range 0 to 255.
- There is a 60% probability that is will generate a random address.

The scanning algorithm selects it's targets as follows: a target host is selected by generating a random IP address of the form $A.B.C.0$, where A, B and C are all random integers in the range 0 to 254. There is a 40% chance, if C is greater than 20, that it will be reduced by a random value between 1 and 20. It will then proceed to scan the entire class C network, incrementing from 0 to 254.

The exploit that Blaster uses differs slightly between Windows 2000 and Windows XP - every scan that takes place has a 20% chance to use the Windows 2000 attack vector, and an 80% chance to use the Windows XP attack vector.

Simulation Programming Specifics

For the Blaster worm, if a worm infection packet arrived at a node, it would generate a random number between 1 and 100 - it would only infect the host if it was a Windows 2000 host (a host was running Windows 70% of the time, and given it was running Windows, it would be Windows 2000 35% of the time) and the number was less than or equal to 20 or a Windows XP host (a host was running Windows 70% of the time, and given it was running Windows, it would be Windows XP 40% of the time) and the number was higher than 20.

Using a Bayesian decision tree, it was then determined that the probability that a scanned machine was an infectable Windows XP host was

$$0.7 \times 0.4 = 0.28$$

and the probability that the host was infectable and ran Windows 2000 was

$$0.7 \times 0.35 = 0.245$$

So when an infection packet was received by the Internet node, it would be a considered a successful infection of the Blaster worm with probability $0.28 + 0.245 = 0.525$. The simulation of the Blaster worm is detailed in Section 6.6.4.

6.4.5 Welchia

The Welchia worm was devised to counter the Blaster worm. Documented in Perriot (2008), it was first discovered on August 18, 2003, and uses a variety of exploits to gain access to Windows 2000 and Windows XP hosts, most notably the same exploits used by the Blaster worm (see Microsoft (2003a) and Microsoft (2003b)).

When Welchia infects a host, it deletes any trace of the Blaster worm from the host's hard drive and memory. It then proceeds to scan for further infections - its scanning algorithm (explained in Perriot (2008)) chooses one of the following two patterns for infection:

- Consider the current host to have IP address $A.B.C.D$ - Welchia then incrementally scans $A.B.0.0$ through to $A.B.255.255$
- Selects a random IP address

Once a target has been determined, Welchia continues by exploiting either the same vulnerability that Blaster uses (and leaves open), or a vulnerability in WebDAV, a component in a webserver.

As Symantec does not state the probability of the scanning algorithms, various values have been assumed. From simulation experiments performed in this research, some success has been found with a 2% incremental scan probability and a 98% random scanning probability.

Programming Specifics

The Welchia worm used the same DCOM RPC exploit as Blaster did, with the inclusion of an additional exploit in WebDAV (a component of the IIS web server). The IIS/WebDAV exploit was much more targeted than the Windows RPC exploit, and as a result would, using the parameters specified above, only infect a small proportion more of the Internet than Blaster. The simulation of the Welchia worm is detailed in Section 6.6.5.

6.5 Simulations

In this chapter, the simulator that was designed and detailed in the previous chapter is rigorously tested through a variety of simulations. The results of these simulations are then compared to some metric (depending on the subject of the simulation this could be captured data, a conceptual "common-sense" model, or an analytical model), and the difference evaluated in the next chapter.

The format used for these simulations remains the same as that used in the previous chapter, as described in Section 5.3.

6.5.1 Statement of Results

The results of the simulation will be stated, and shown in summary. Where the design of a simulation is not immediately apparent, diagrams have been used to display the configuration and execution of the simulation.

Hilbert Curves

A Hilbert curve (or "Hilbert Space-Filling Curve") is a novel way to efficiently represent a large quantity of 1-dimensional data using a 2-dimensional fractal curve. In Irwin and Pilkington (2008), the authors describe a Hilbert curve as "a continuous fractal curve, the limit of which fills a square." The important property of the curve that is relevant to IP space visualition according to the aforementioned authors, is that it "maintains locality of data on the curve - this means that data ordered a certain way in one dimension will still be ordered the same way along the curve in two dimensions." (Irwin and Pilkington (2008)).

The Hilbert curves used here allow an ordered visualisation of all of IP space (with some loss of detail due to the high resolution required) by fractally dividing a square image into blocks of available IP addresses (ordered by significance of byte in the IP address), with the lowest values starting from the top left to the highest values in the top right.

An example of a Hilbert curve representing the first octet of an IP address can be seen in Figure 6.1, as shown in Irwin and Pilkington (2008).

The “order” of the curve is the number of fractal iterations it has undergone, which (for a two-dimensional Hilbert curve) multiplies the number of “points” by four per increased order. A first order Hilbert curve has 4 points - the 4^{th}, 8^{th}, 12^{th} and 16^{th} order curves have numbers of points corresponding to the number of class A, B and C networks and individual hosts on the Internet respectively. which further adds to their suaitability for use in IP network address visualisation.

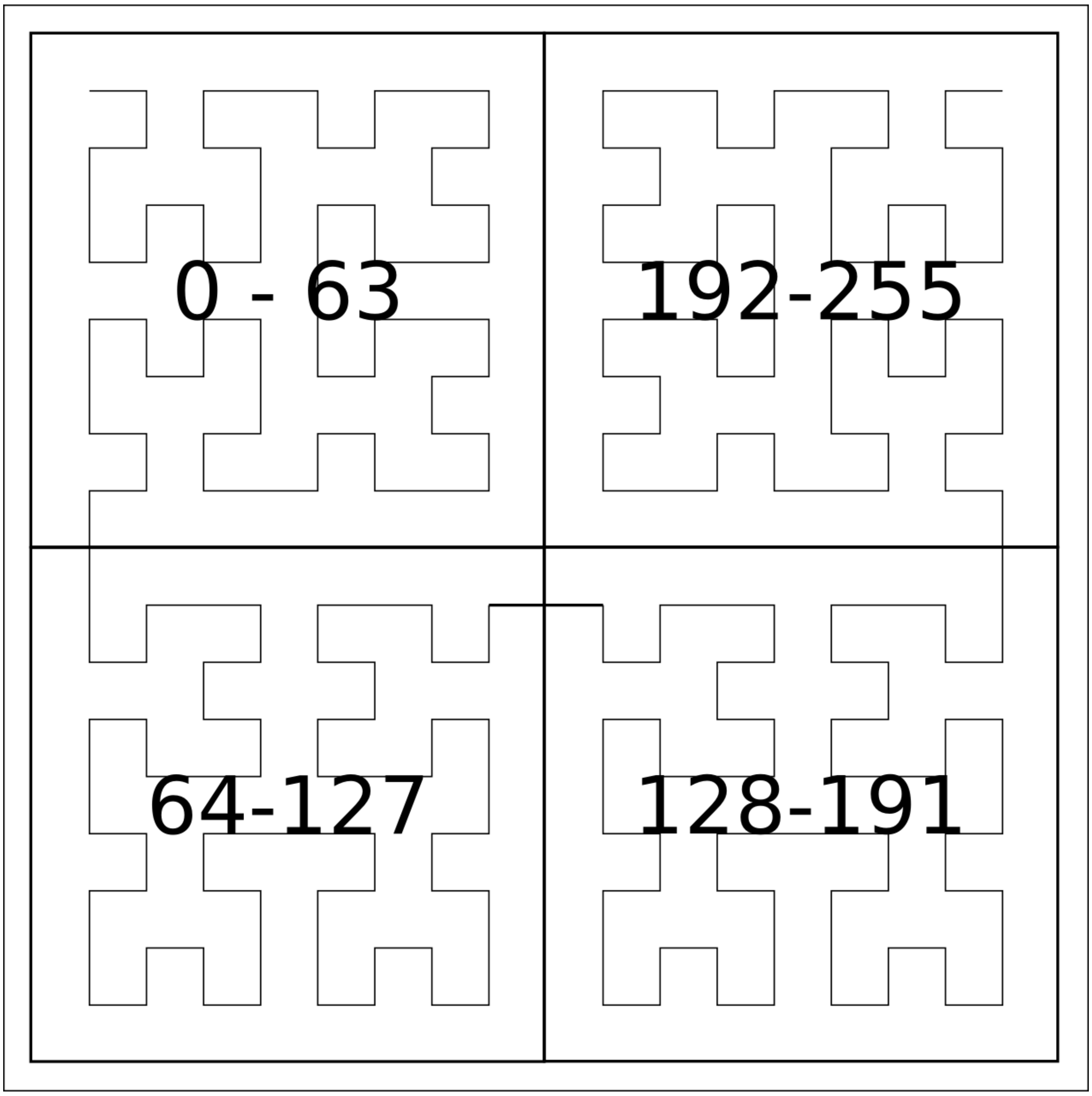

Figure 6.1: Order 4 Hilbert Curve

Thus, the curve shown in Figure 6.1 can then be further divided for the next most significant octet (which will be order 8) and so on. For the purposes of illustration in this document, Hilbert curve visualisation will be limited to order 12 (each point on the curve representing a group of 256 hosts which provides a logical mapping to a network of size /24 in CIDR notation or a traditional Class C network) instead of individually rendering each host, as the resultant curves would be too large (and the points too small) for easy study. Where appropriate, further Hilbert curves with magnification are also provided for explanation.

6.6 Worm Testing

As discussed in Section 4.2, Internet worms are an appropriate choice for subject simulation, as they put stress on a simulation system and are well-documented for comparison. As a result, several worms were used to test the developed simulation system. The worms chosen for simulation were selected on the basis of documentation and scaling complexity: those worms which had scanning algorithms, exploits and other information published allowed for better simulation modeling opportunities. The Witty worm was selected due to its simplicity, while the Blaster and Welchia worms were selected due to their complexity in contrast to the Witty worm.

6.6.1 Simple Worm

The simple worm case uses a worm with 100% infection probability, which randomly scans across the entire IPv4 address space. While simplistic, this worm provides a suitable starting point for the development of more complex worm simulations as it already addresses the initial problems: generating packets in exponentially-growing quantities, infections affecting certain specific nodes, and so on.

Statement of Subject

The scenario to be simulated can be described as follows: a single host node in the network detailed above (in simulation 6, Section 5.4.7) has been infected with a Simple Worm. Simple Worms have a perfect infection rate (so every host that receives a packet will immediately be infected, regardless of operating system, software, Internet firewalls, etc.), and they spread by randomly scanning the Internet. The Simple Worm also scans quickly: every simulated "tick" or time-period in the simulation will result in every host that has been infected randomly infecting another host.

While this is an unrealistic example, it serves to provide a framework for all further worm simulation: functions for scanning algorithms, likelihood of infection, and seperate behaviours for host and network nodes will already be in place and can easily be adapted for parameters of "real worms" for rapid development of real worm simulations.

Statement of Parameters

The network as described above should be reconstructed (sixteen mesh-connected host nodes with a router passing non-local traffic to a network node), and one randomly determined node should be specified as “infected”.

The worm simulation should run for a large number of cycles (at least a million ticks), and the results should then be documented. Due to the random scanning nature of the worm, and the small size of the local network, it is unlikely that the worm will scan a host node. Behaviours should be in place for this eventuality, and if a host node is infected, it should immediately join the initial randomly chosen node in creating new infection packets to send out.

Statement of Results

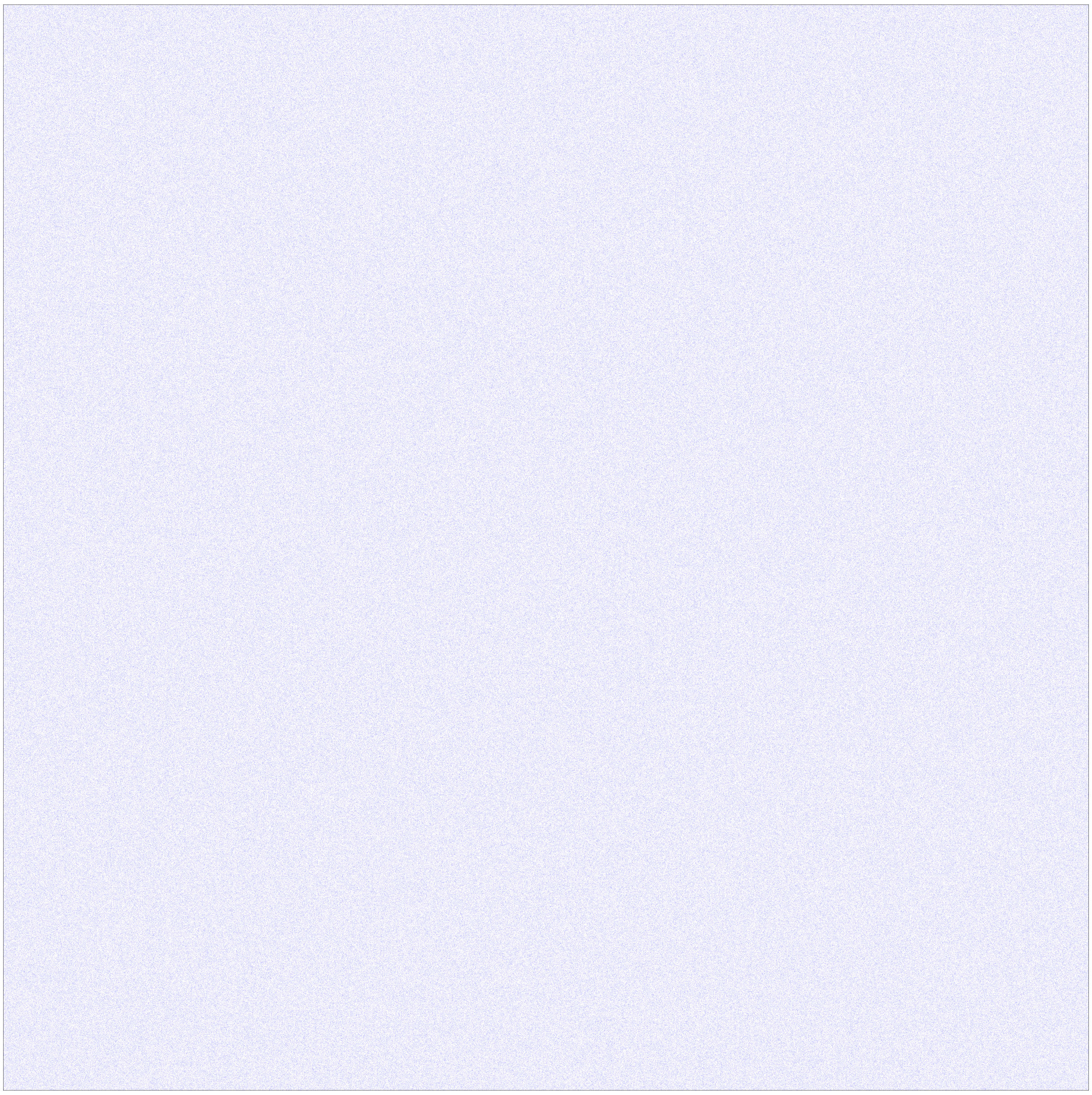

Figure 6.2: Simple Worm Hilbert Curve

The Simple Worm, executing 1 000 000 ticks (implying 1 000 000 uniquely scanned hosts) resulted in 1 000 000 infections distributed randomly. Note that only one million ticks were simulated (most other simulations using ten times the number of ticks), as the simulation became untenable due to the massive rate of infections, combined with the perfect infection rate. The results are summarised in a Hilbert Curve, Figure 6.2.

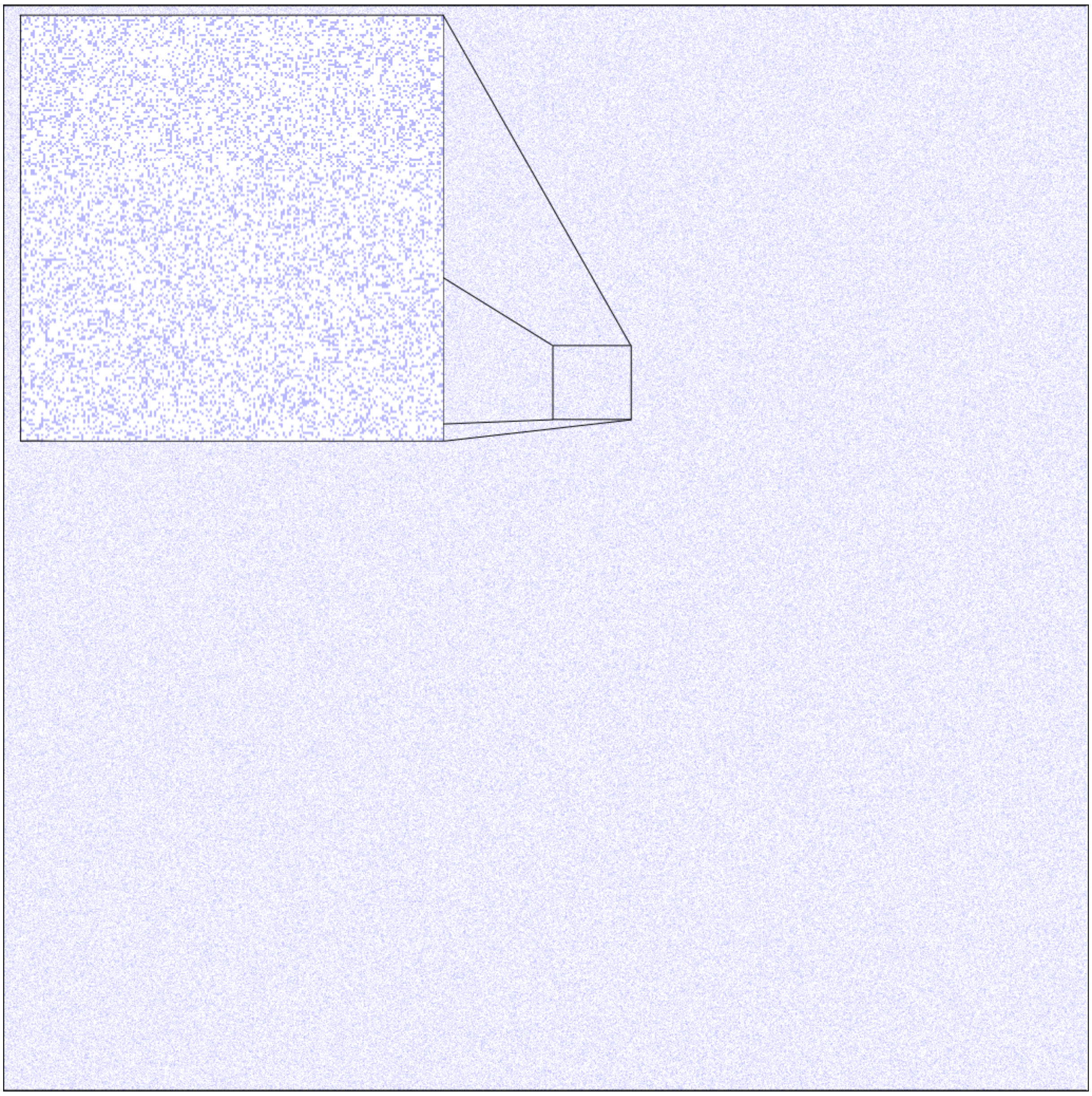

Figure 6.3: Simple Worm Hilbert with Magnification

The Simple worm results are as expected - the worm has scattered values randomly but uniformly across the entirety of IP space, with a very high density due to the perfect infection rate and very high scan rate. They can be seen visualised and magnified in Figure 6.3.

6.6.2 Witty Worm

The Witty worm is mostly an extension of the simple worm, as it is also a random scanning worm. Significant differences between the worms can be noted in the infection rate (Witty worm has several very specific requirements for systems which it can infect, and as such has a much lower infection rate per packet sent out), but a much higher rate of packet sending, to represent the multi-threaded, small-packet way in which it is spread.

Statement of Subject

The Witty worm is (in terms of parameters) very similar to the Simple Worm described in simulation 7 (Section 6.6.1). It also uses random scanning, and also has a very fast scanning rate.

The significant difference is in the infection rate. As described in the previous chapter (Section 6.6.2), the Witty worm has a low rate of infection (requiring both a Windows host, and the ISS software to exploit) of 3.5%. This is represented in 96.5% of packets that are received by the network node being "dropped" instead of logged, as they would have been ineffectual in infection.

Statement of Parameters

The network described and used in Section 5.4.6 will suffice for this simulation, and the only significant difference to be implemented between these two worms is a much smaller infection rate.

Again, only one host will be infected. This host will then proceed to attempt to infect other hosts, but the likelihood of local infection (and thus an increase in scanning rate) is low due to the random scanning algorithm.

Statement of Results

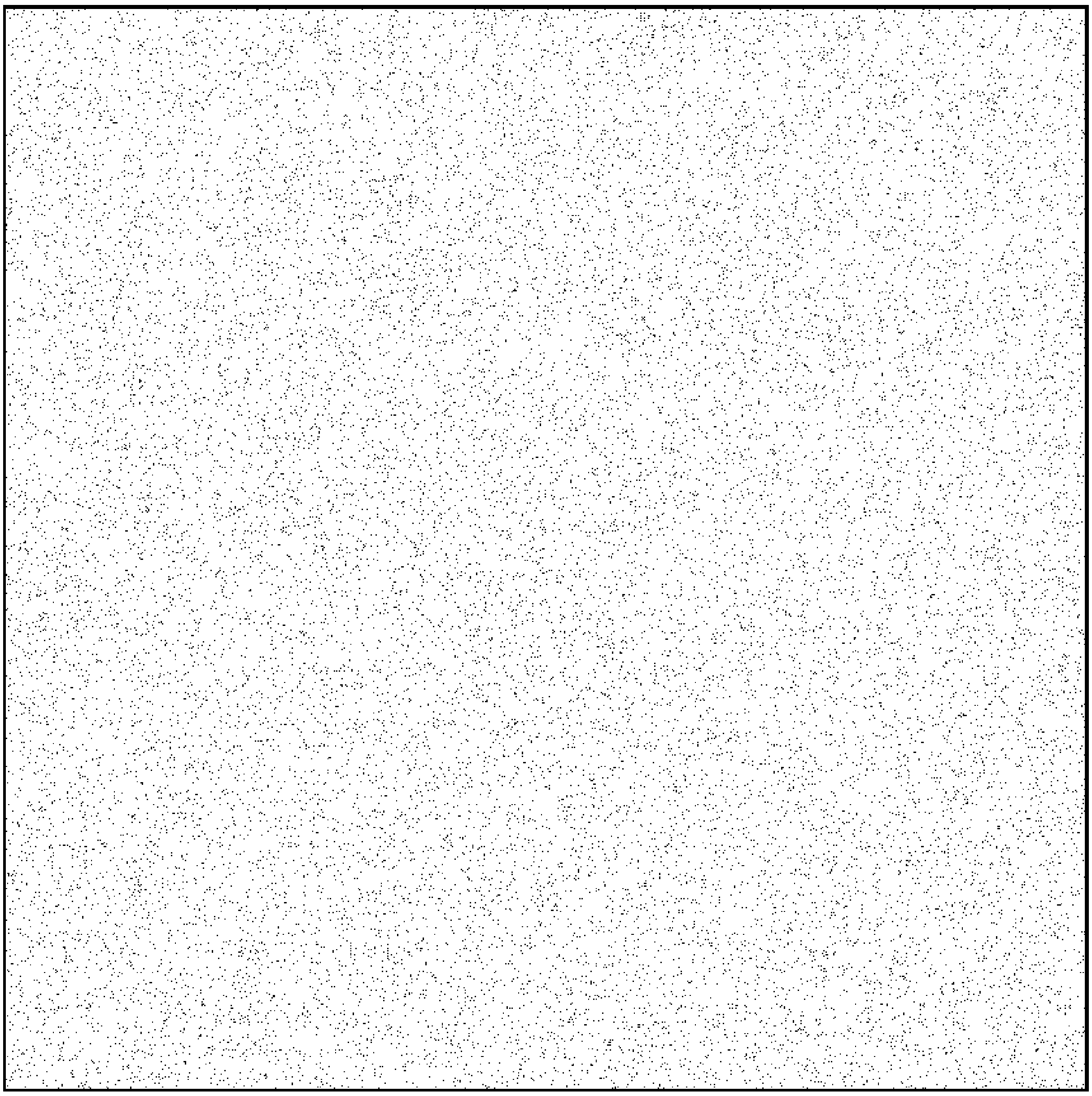
Figure 6.4: Witty Infection Hilbert Curve

The Witty Worm, executing 10 000 000 ticks (implying 10 000 000 scanned hosts) resulted in 129 970 infections distributed randomly.

The results are summarised in a Hilbert Curve, Figure 6.4.

The Witty worm is similar in all ways to the Simple worm (in that it randomly scans uniformly across IP space, very rapidly), with an obvious lack of density due to the comparatively low infection rate. As a result, no magnification of the results of this simulation is necessary.

Commentary

The Witty worm scanned randomly - so the expected results were a randomly distributed series of scans. Figure 6.4 confirms that the simulator was effective. Analysis of the outputs reveals that the highest number of scans on a class C network was 3, and the average number of scans per class C network was 1.00404 - indicating a very broad, but very "shallow" random scan.

6.6.3 Code Red vII

The Code Red vII worm is the first simulated worm with a more intelligent scanning algorithm. While it has a higher infection likelihood than Witty, it specifically avoids certain IP blocks (127.0.0.0/8 and 224.0.0.0/8 specifically, as these are special IANA reserved networks - localhost and multicast networks respectively - documented in (IANA)), and is more likely to scan locally than random scanning, as shown in the work done by Friedl (2001).

Statement of Subject

The Code Red vII worm is the first worm that will be simulated with a "scanning algorithm" or pattern. It has a higher likelihood of scanning conceptually "nearby" hosts than the Witty or Simple worms. It also exploits software more common than Witty's (Code Red vII used an exploit in the Microsoft IIS web server, as documented in Microsoft (2003c)), resulting in a much higher infection rate of 21%, but distributed a much larger binary, and as a result transfer speeds were longer than Witty.

Statement of Parameters

The Code Red vII scanning algorithm was implemented as the real worm was programmed: the worm chose a random value between 0 and 7, then used the following decision-making to determine the addresses to scan:

- if the value was 0, it would choose a random IP address to scan
- if the value was greater than 0 and less than 4, it would use the first byte of the sending host in the scanning address and the remainder of the IP was randomly generated
- if the value was 4 or greater, then the first two bytes of the sending host were used in the scanning address and the remainder of the IP was randomly generated

To represent the higher infection rate, only 79% of the Code Red vII worms' packets that are received by the network node are ignored. Finally, to represent the (relative) lack of speed in the spread of the worm due to bandwidth limitations, the time between a node sending a worm packet and its arrival (and therefore its infections) is increased from a single tick to a random number of ticks between 5 and 7.

Statement of Results

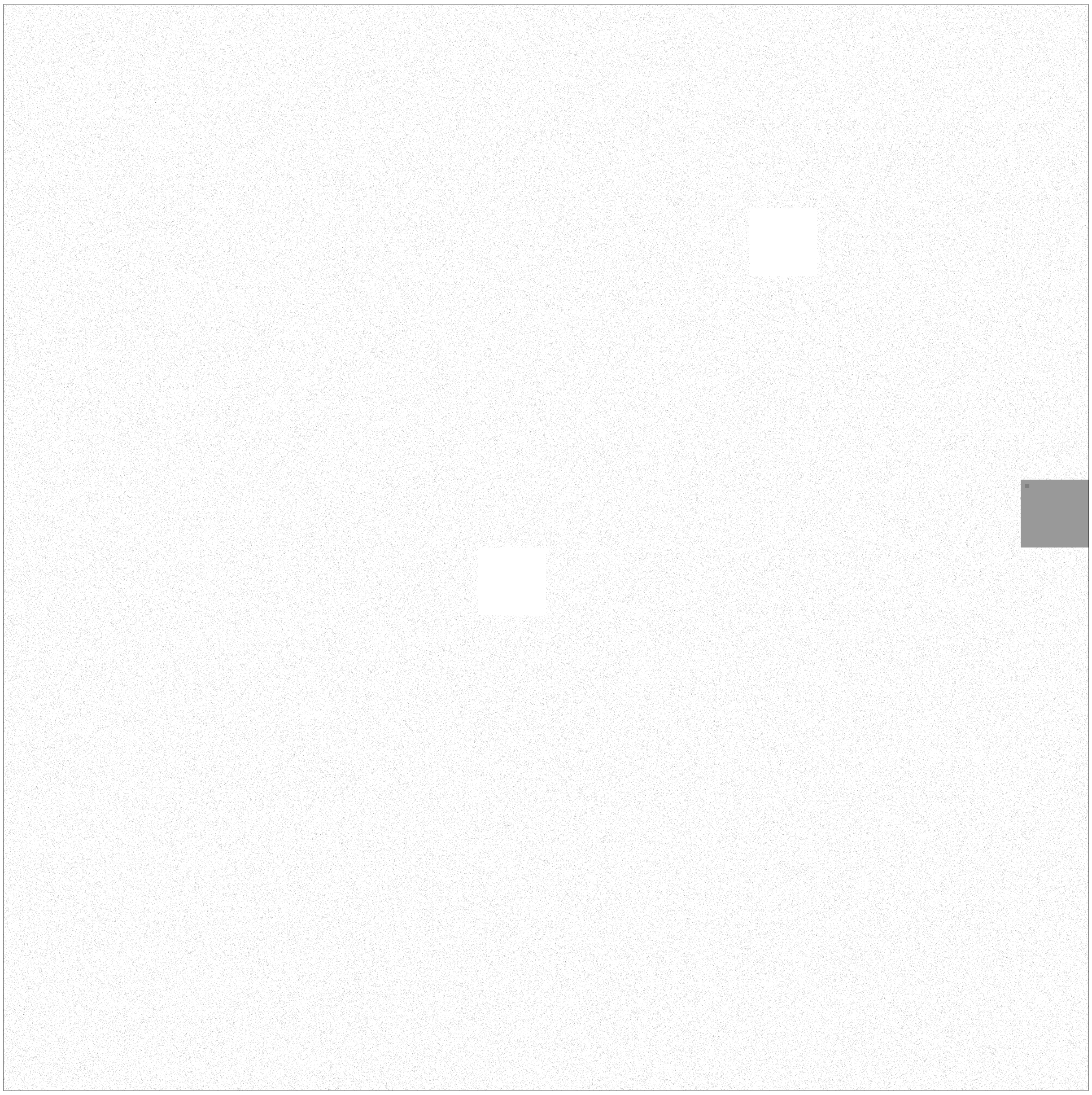

Figure 6.5: Code Red vII Infection Hilbert Curve

The Code Red vII Worm, executing 100000 ticks (implying approximately 17000 uniquely scanned hosts) resulted in 4016 infections clearly clustered around the networks nearby to the infecting computer (the 192.168.0.0/16 network, in this case). The results are summarised in a Hilbert Curve, Figure 6.5.

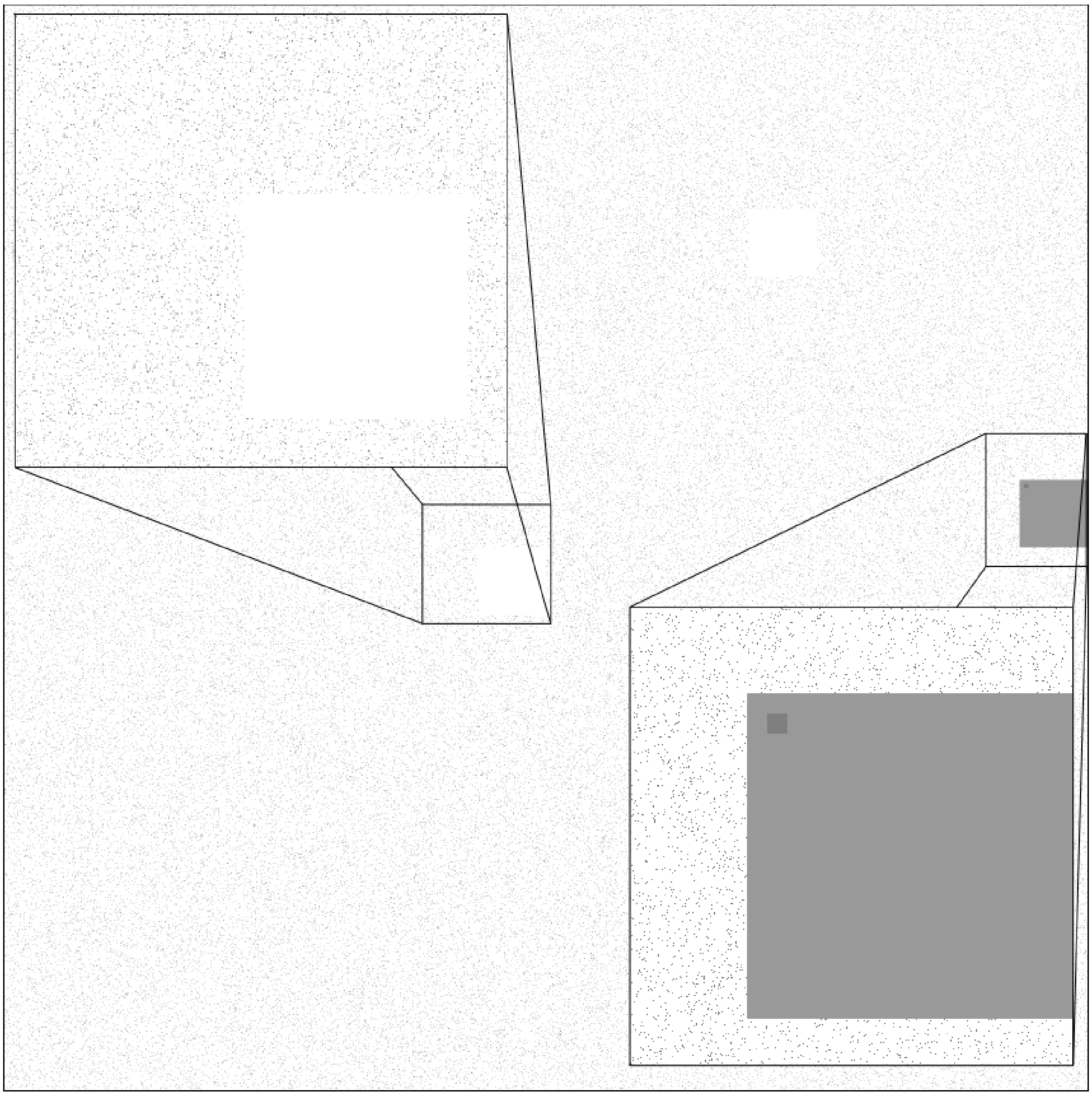

Figure 6.6: Code Red vII Hilbert with Magnification

The Code Red vII worm shows the results of the scanning algorithm presented here. Referring to the magnification of the visualisation of Simulation 9's results (Figure 6.6), it can clearly be seen that the local class A network has been preferred, with a focus on the local class B network, which can be seen as the darkened portion of the class A network.

The other immediately noticeable part of the results is the lack of any activity in the 127.0.0.0/8 and 224.0.0.0/8 networks, as the scanning algorithm of the Code Red vII worm specifically avoided both. This is seen as the large white blocks in the Hilbert curve.

Commentary

The Code Red vII worm was designed with three scanning modes that it chose randomly: local (class C), local (class B) and entirely random. Thus, the expected output was an intense, very focused local scan, a less intense loosely focused local scan and a random scanning pattern otherwise, with the exception of two specific IP blocks: the 127.0.0.0/8 network and the 224.0.0.0/8 network. Figure 6.6 confirms that the simulator has successfully simulated this.

6.6.4 Blaster

The Blaster worm, like the Code Red vII worm, has a higher infection rate than the Witty worm, and also prefers local to global scanning.

Statement of Subject

The Blaster worm simulation uses a conceptually simpler but technically more advanced scanning algorithm than the Code Red vII worm: it picks a random class C to begin scanning, and completely scans all 254 IP addresses before randomly selecting another network.

The Blaster worm was another (relatively) large worm that had to use multiple packets for infection (thus slowing down propagation), but also had a very high infection rate (because of the prevalence of Windows computers, the infection rate was approximately 52.5%).

Statement of Parameters

The Blaster scanning algorithm operates as described above, with one additional modification: in 40% of cases where the third byte of the IP address was greater than 20, the value would be reduced by 20. Because many networks assign IP addresses in ascending order, this resulted in a greater likelihood of infection. This was taken into account in the scanning algorithm. In a similar manner to the Code Red vII simulation, 47.5% of the Blaster worms' packets that are received by the network node are ignored. To represent the (relative) lack of speed in the spread of the worm due to bandwidth limitations, the time between a node sending a worm packet and its arrival (and therefore its infections) is increased from a single tick to a random number of ticks between 5 and 7.

Statement of Results

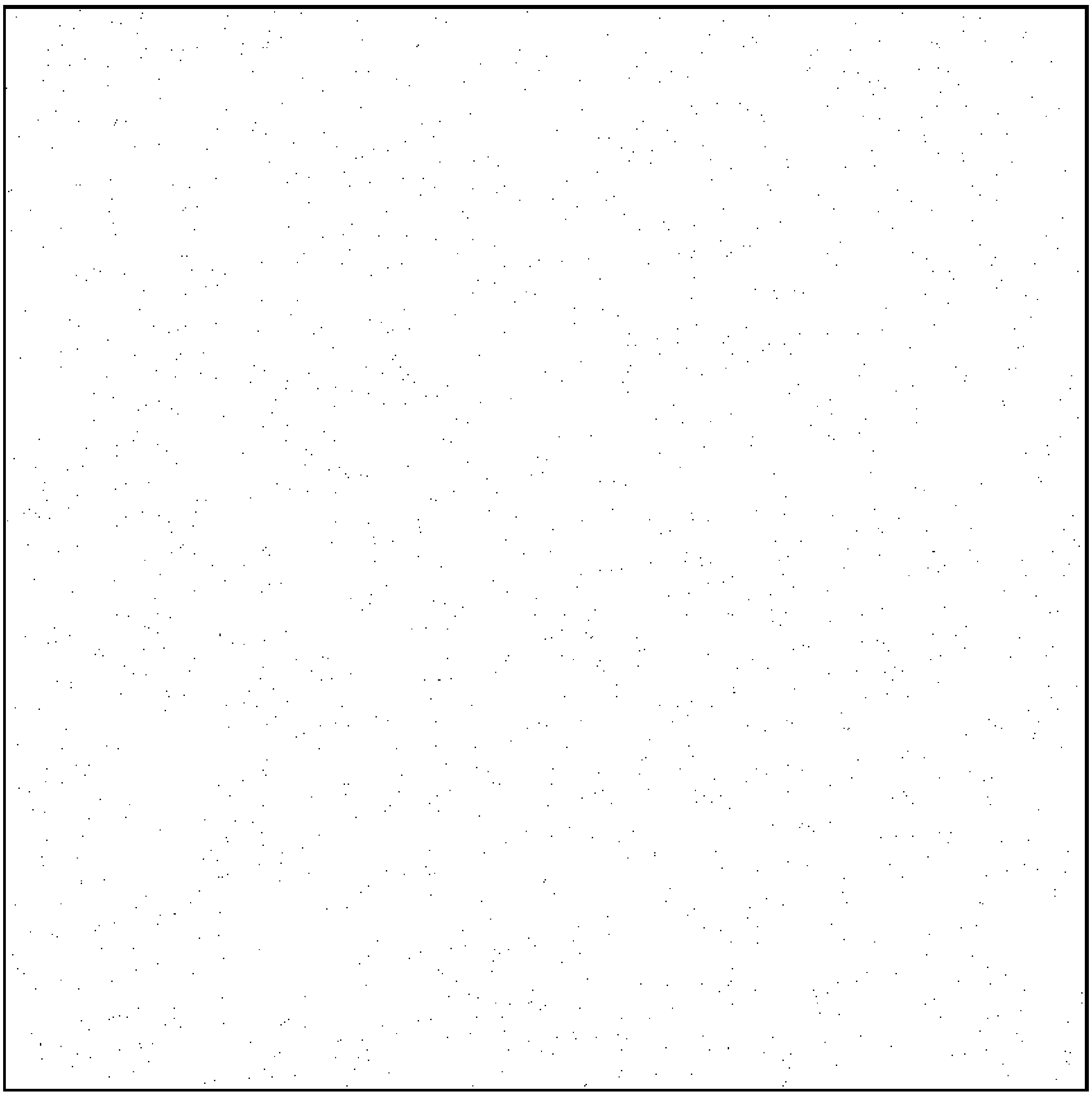

Figure 6.7: Blaster Infection Hilbert Curve

The Blaster Worm, executing 100000 ticks (implying approximately 17000 scanned hosts) resulted in 9772 infections clearly clustered into a few networks, the only observable pixels in the curve. The results are summarised in a Hilbert Curve, Figure 6.7.

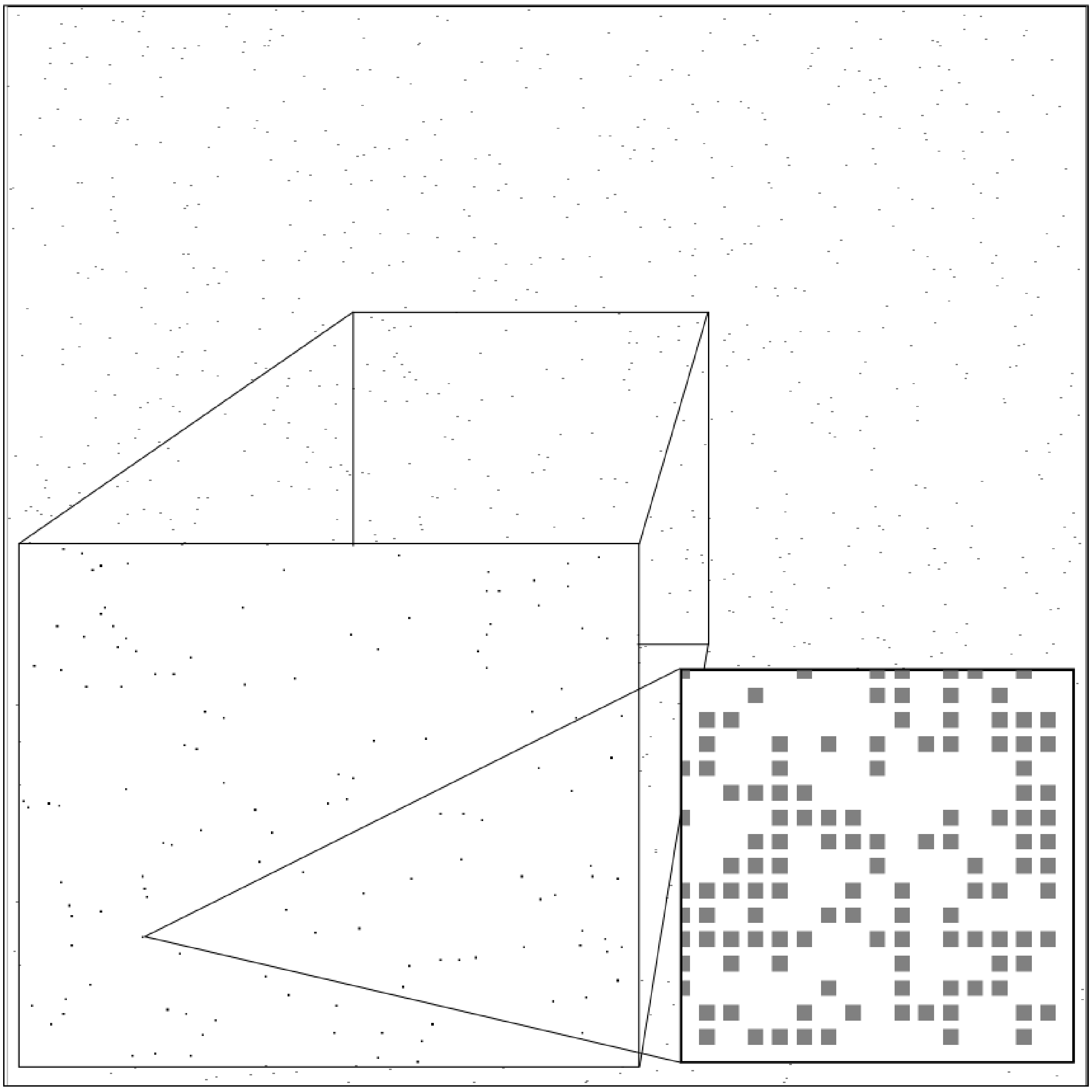

Figure 6.8: Blaster Hilbert with Magnification

The Blaster worm Hilbert curve (Figure 6.8) appears to simply be a random, uniformly distributed series of received packets. This appears so as a result of the granularity of the Hilbert curve - the visualisation cannot show scans deeper than at a Class C level.

Each “dot” on the curve shows a Class C network. If we examine the Blaster worm scanning algorithm discussed in Section 6.4.4, we see that it completely scans a full Class C network. Upon closer examination of the results, it can be seen that each Class C network represented in the Hilbert curve has received many packets compared to the Simple or Witty worms, where the Class C networks received much fewer packets each, but the distribution was much broader.

Commentary

The Blaster worm did a complete scan of a full class C network before randomly selecting another class C network to scan. While Figure 6.7 doesn't completely show this due to scaling problems, the magnification in Figure 6.8 shows that each individual host in the network has been scanned - though only a portion of these are "infected", as the Blaster worm only exploited vulnerabilities in Windows hosts.

Thus, we can conclude that the simulation results are as expected - the Blaster worm appears to select a random class C network to scan and then scans it completely, infecting those nodes which would be vulnerable.

6.6.5 Welchia

The Welchia worm, while technically lacking a purpose (and a large body of infectable hosts) without the Blaster worm, is another worm with a local-scanning priority and good infection likelihood.

Statement of Subject

The Welchia worm was designed to counter the Blaster worm, and as a result normally exploits a weakness that the Blaster worm exposes in systems that it infects. As a result, the simulation of a Welchia worm without a Blaster worm relies on the other form of infection that it used: a vulnerability in the WebDAV component of IIS version 5.0. This results in a much lower infection rate than if Blaster-compromised hosts were available for the simulation.

The Welchia scanning algorithm operates similarly to Blaster, except that it scans entire class B networks (i.e. 65536 hosts instead of 256). Otherwise, it is similar in most respects to the Blaster worm.

Statement of Parameters

The Welchia worm will use the same simulation network as the other worm examples, but operates by randomly selecting a class B network and completely scanning it: 256 networks of 256 hosts results in a very thorough network scan.

The Welchia worm ran multiple exploits in an attempt to gain access to the system. This includes the DCOM RPC exploit used by the Blaster worm, referenced in Section 6.6.4 above. The other exploit it made use of was the WebDAV vulnerability in a specific version of IIS.

This series of simulations assume that Windows hosts account for 60% of the Internet, and that 2% of those hosts have the correct version of IIS with exploitable WebDAV installed, resulting in a 1.2% infection rate assuming that Blaster is not a factor.

Thus, the 52.5% that is calculated above (from the DCOM RPC vulnerability) combined with the 0.57%[1] representing the still-vulnerable hosts with the WebDAV vulnerability result in a 53.02% infection chance.

Statement of Results

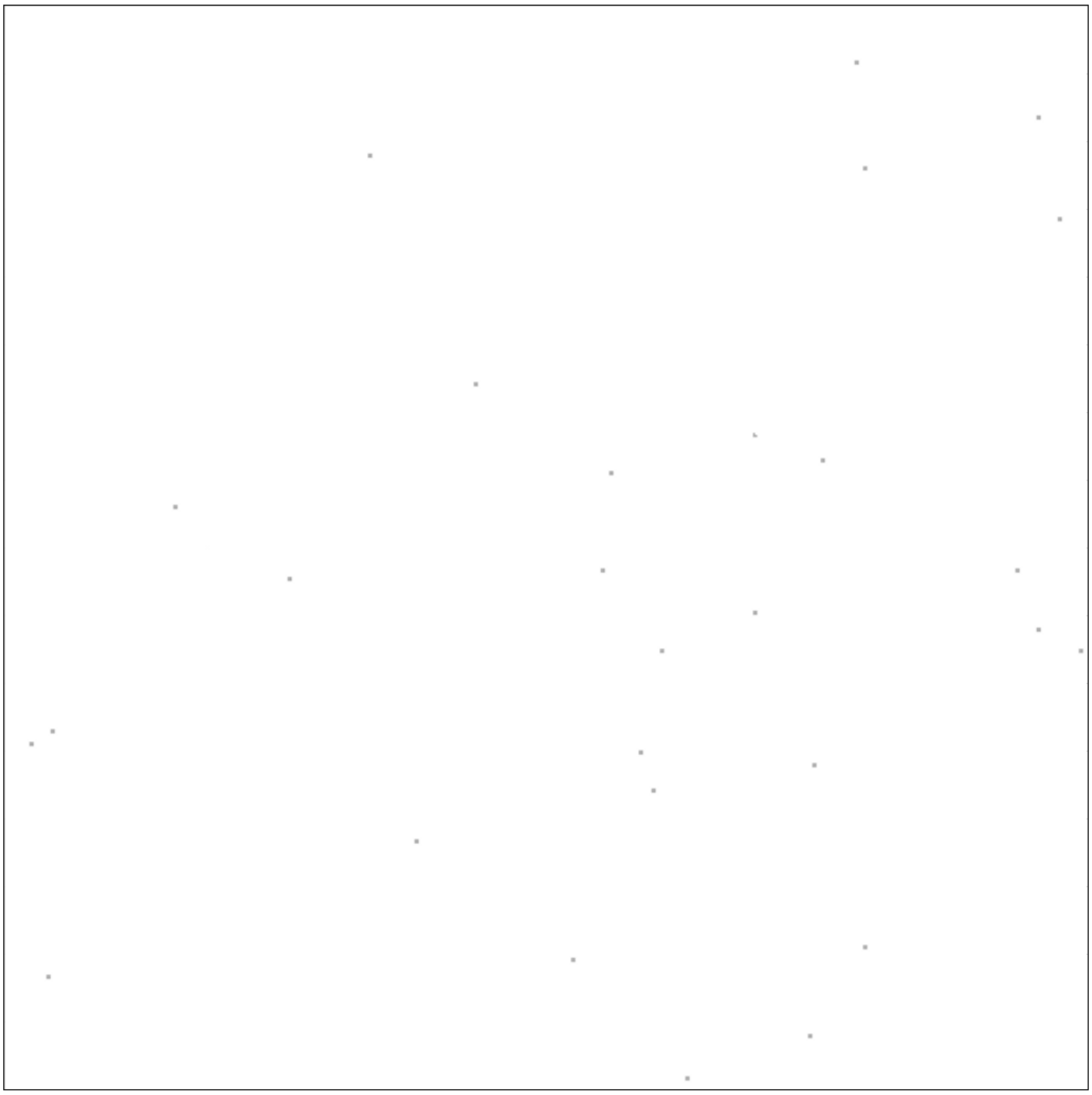

Figure 6.9: Welchia Infection Hilbert Curve

The Welchia Worm, executing 100000 ticks (implying 17000 uniquely scanned hosts) resulted in 238 infections (much less than the rest), all located in the same class B

[1]The remaining unexploited hosts represent 47.5% of hosts, with a 1.2% chance of infection: $0.475 \times 0.012 = 0.0057$

network - the dots on the curve represent approximately 64 class C subnetworks in the 190.31.0.0/16 network. The results are summarised in a Hilbert Curve, Figure 6.9.

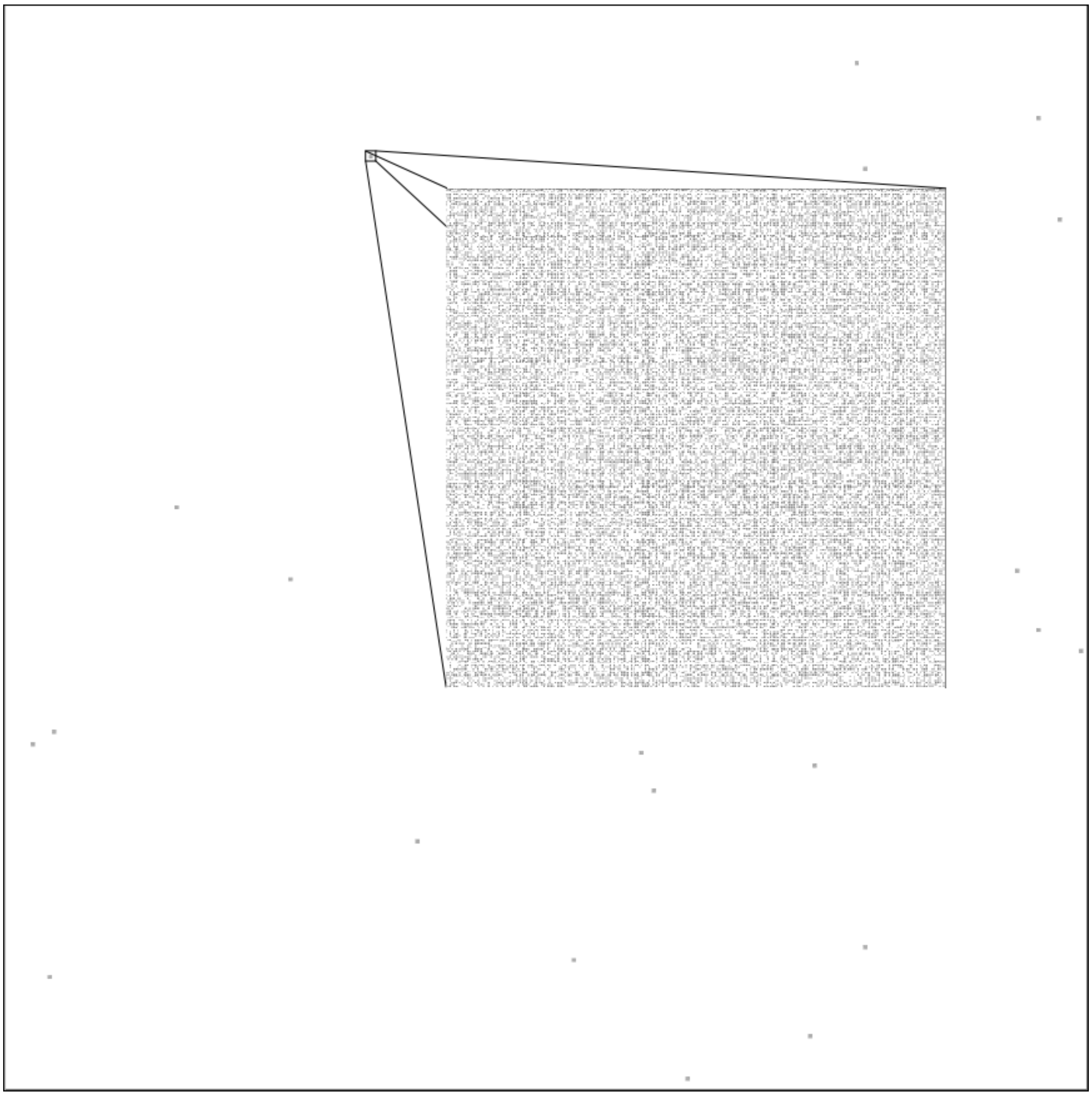
Figure 6.10: Welchia Hilbert with Magnification

Contrasted to the Blaster results from the above section, the magnified Welchia results (Figure 6.10) show the Welchia worm's results with much greater clarity. The Welchia worm's scanning algorithm (see Section 6.4.5) show that it scans complete Class B networks - comprised of a block of 256 Class C networks - and with magnification, it becomes evident that this is what has occurred. By looking carefully at what might otherwise appear to be large "dots", we can see that the Welchia worm scans the Class B network, but each Class C network within it has different degrees of infection. This corresponds with the expected results.

Commentary

The Welchia worm completely scanned complete class B networks, which it chose at random. This leads to the expectation that the results of simulating the Welchia worm would be a few, randomly distributed networks which would be intensely scanned.

Figure 6.9 supports this - we can see that the Welchia worm has scanned uniformly and randomly. This is further confirmed in Figure 6.10, where magnification shows that the "dots" where the worm scanned can be seen as thoroughly scanned. In a similar manner to the other "real" worms in this chapter, it does not scan perfectly, as it requires a certain software configuration to be effective. This effect is seen by the white areas in the magnified curve.

6.7 Advanced Simulations

In this section, advanced simulations which are beyond the scope of Internet worms are documented. These advanced simulations have been chosen to demonstrate the practical value of this simulator - the concepts presented here are all valuable tools in the field of network security, and by testing the effectiveness of these tools, the simulator can be used as a fully-fledged research tool.

6.7.1 Red vs. Blue (Simple vs. Simple)

The concept of Red vs. Blue worms pertains to the "fight fire with fire" strikeback concept of worm countermeasures. By creating a worm that uses the security holes created by another worm, deleting the malicious worm, spreading and then deleting itself, it hopes (conceptually) to work as a form of defence against malware.

Initial testing of the concept can use a pair of simple worms. Random scanning means that there is a very low chance of collision between these two worms, but the simpler implementation allows for more rapid testing of the concept.

Statement of Subject

The purpose of this simulation is to test the viability of a worm vs. worm concept. Meaningful results are not expected, however in the following simulation a more interesting and practical simulation will use the principles developed in this one.

The core subject of this simulation is a pair of lists of infected hosts, and specifically noteworthy are any "collisions" in the lists - where an entry appears in both, the worms have intersected. This would imply that the "blue" or helpful worm (helpful worms are considered in Section 4.2.4) is effective in removing an infection from a host.

Statement of Parameters

In this test, two nodes are infected with perfect random scanning worms (similar to that used in Section 6.6.1, Simulation 7), each marked with a different "colour", which is also specified in the packets they send. They are connected to a small local network (16 nodes) with an attached network node. These nodes then proceed to send packets which are collected by the network node, and are sorted into one of two infection lists, depending on the worm's type.

At the end of the simulation, both worms are listed and any collisions are noted.

Statement of Results

The simulation completed and no worm intersection/collision occurred at all, over the course of ten million ticks (approximately 9.9 million infection cycles for each worm). A very large portion of the Internet was scanned by both worms (approximately 25% of all Class C networks experienced a scan from each worm).

Commentary

This first "Red vs. Blue" example was developed to prototype the concept - the expected results were few collisions, with general random scanning, with no useful results beyond testing the operation of the simulation. This was confirmed - no collisions occurred, and the simulation did not produce any noticeable patterns.

6.7.2 Red vs. Blue (Blaster vs. Welchia)

The classic Red vs. Blue worm case, the Welchia worm was released "into the wild" in order to combat the Blaster worm. By exploiting a "back door" that the Blaster left open, the Welchia worm rapidly spread across the Internet.

Statement of Subject

This simulation attempts to recreate the Blaster and Welchia worm scenario. The core focus of this simulation is on the effectiveness of the Welchia worm in finding Blaster-infected hosts, and to determine whether its effectiveness as a worm counter-measure mitigates the possible problems that unleashing a worm presents.

Statement of Parameters

In a similar setup to the parameters used in Section 6.7.1, two nodes on a network are infected with Blaster and Welchia worms and ten million ticks are executed by the simulation engine. The significant difference is the inclusion of non-perfect infection rates (corresponding to the values considered in Section 5.2.2) and the worm-specific scanning algorithms.

Statement of Results

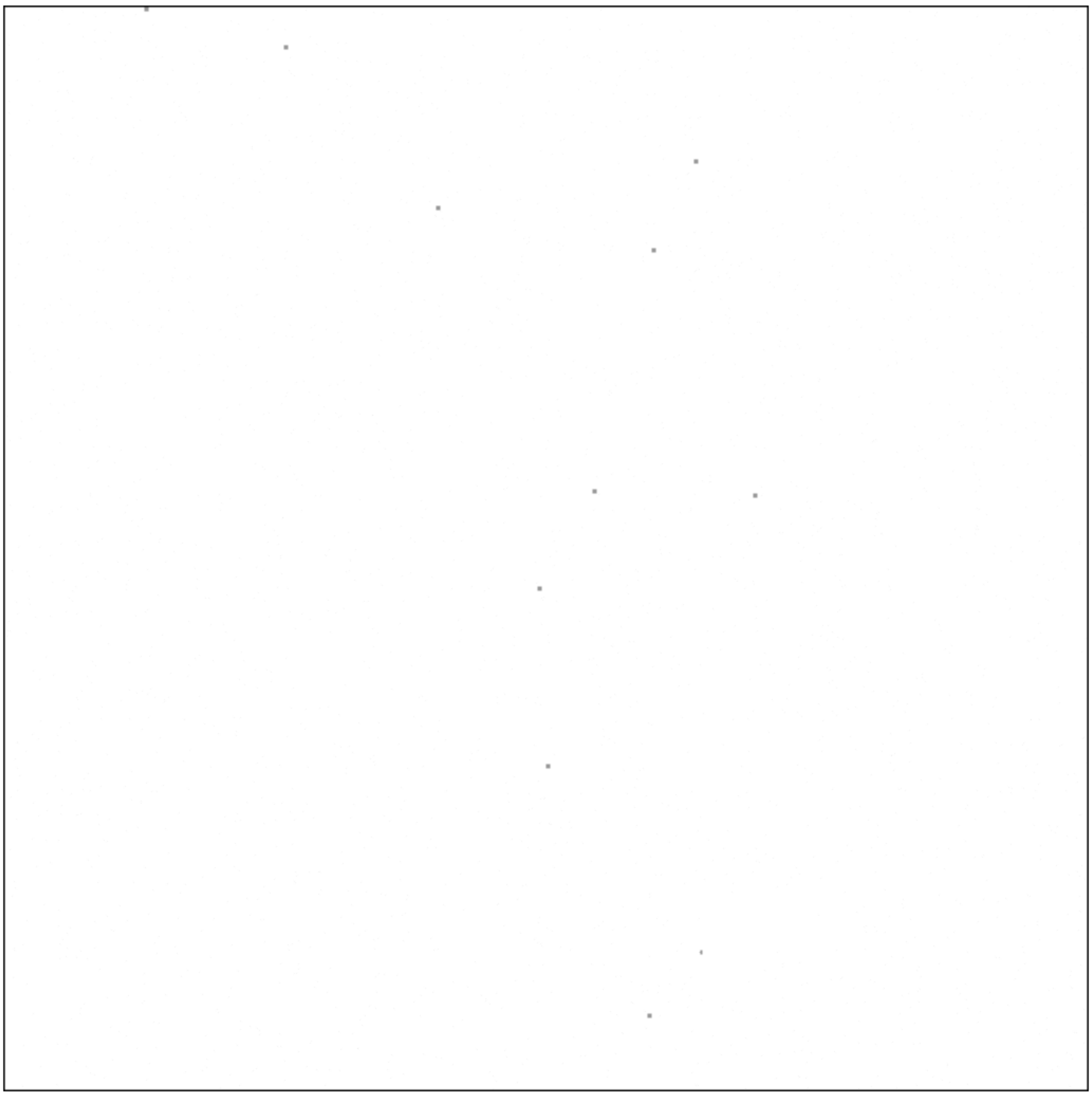

Figure 6.11: Red vs. Blue Worm Hilbert Curve

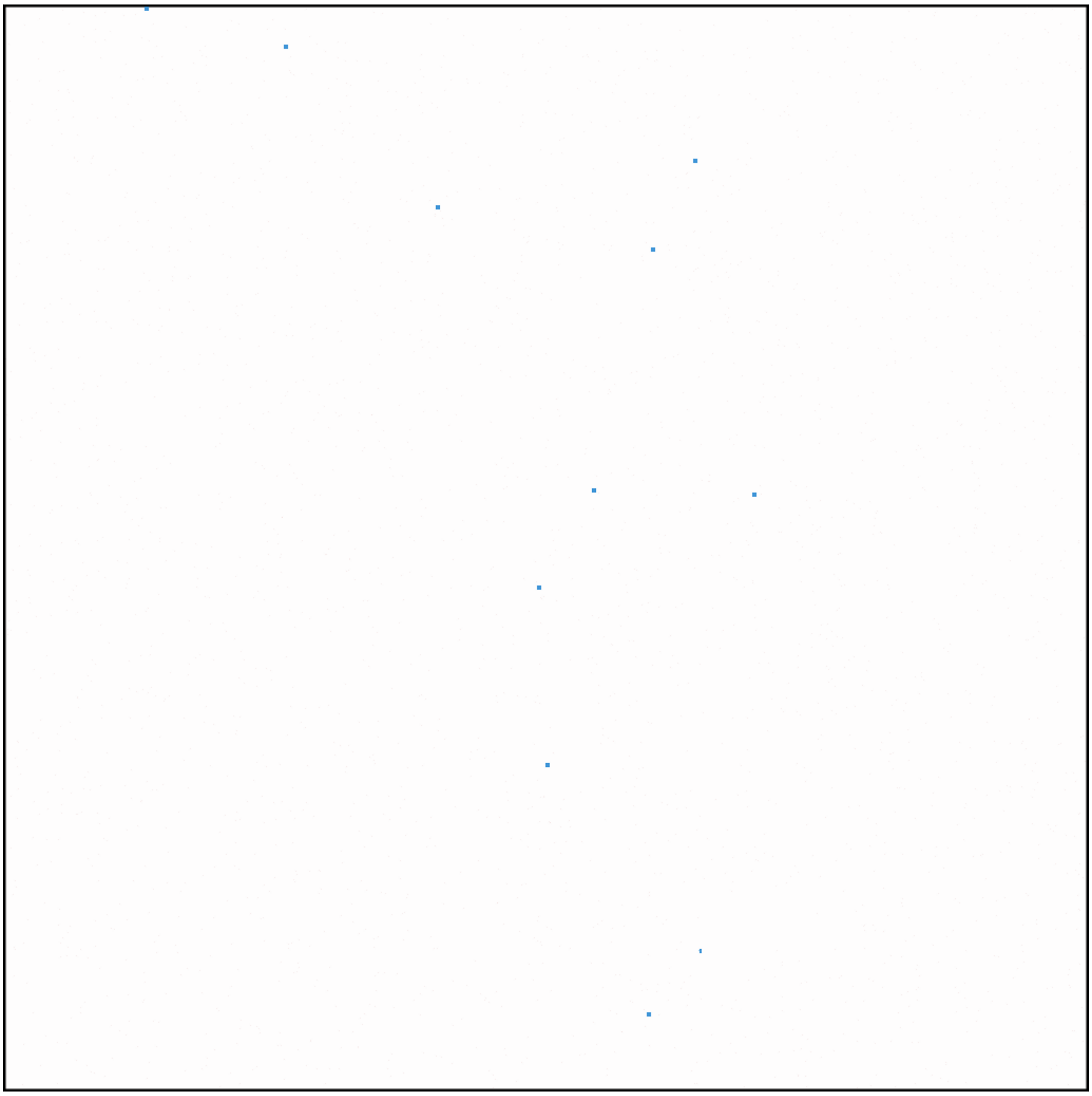

Figure 6.12: Colour Red vs. Blue Worm Hilbert Curve

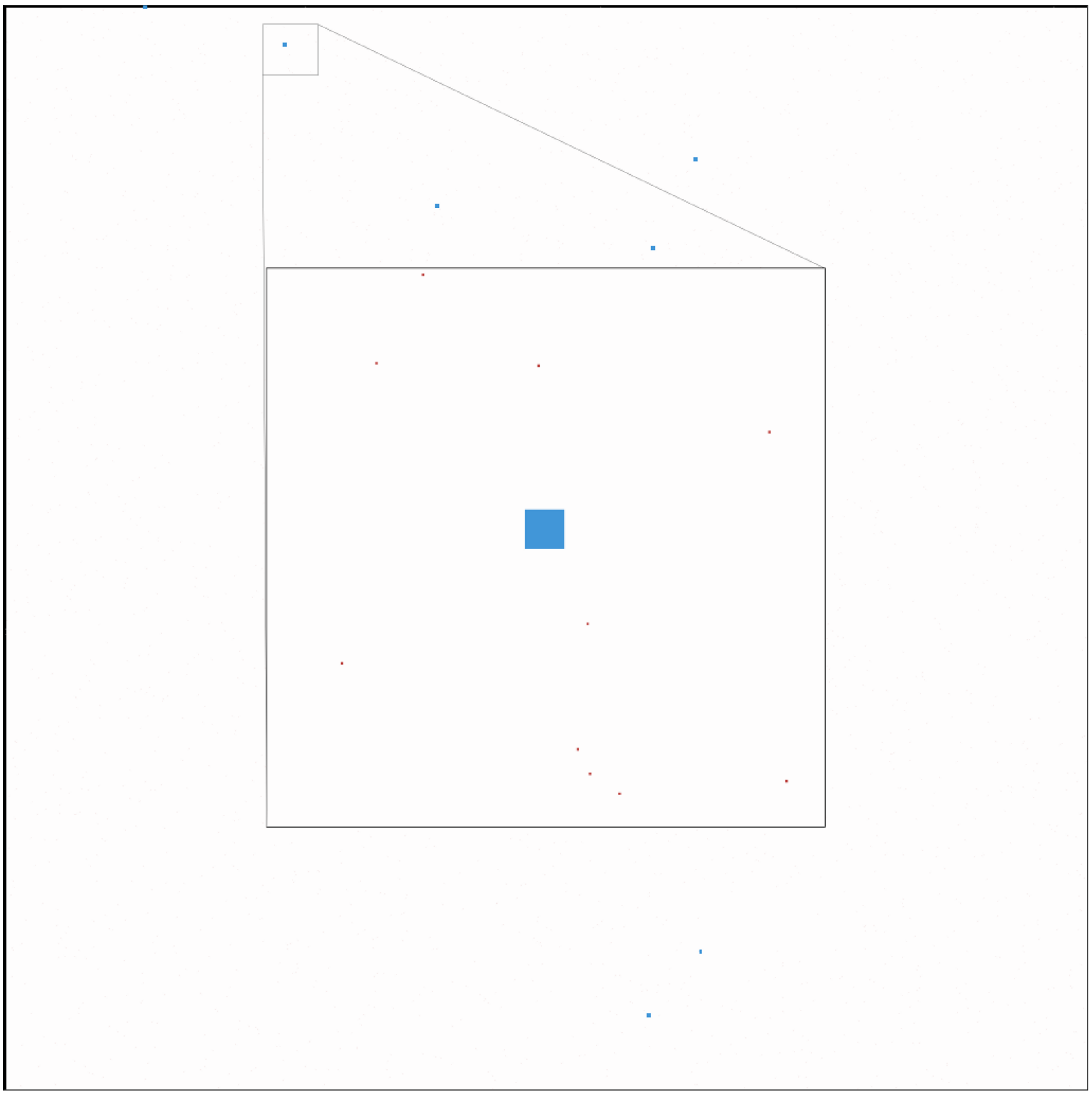

Figure 6.13: Colour Red vs. Blue Worm Hilbert Curve with Magnification

The visualised results in Figure 6.11 are similar to a union of the Blaster and Welchia worms simulated in Sections 6.6.4 and 6.6.5 respectively (as expected). Figures 6.12 and 6.13 show the results in colour, with Figure 6.13 magnifying a particular portion of the network for further study.

One set of collisions occurred over the entire ten million tick simulation.

Commentary

The applied Blaster Red vs. Welchia Blue scenario presented in this simulation is based on the Simple worm case developed in the previous simulation with extensions for the two worms.

It was expected that the Blaster worm and Welchia worm would operate as they did in the simulations where they were used in isolation, but in the case of collision, a high number of collisions would occur in a very small number of ticks. Because the Blaster and Welchia worms both scan entire networks (class C and class B network respectively), if a collision were to occur, the entire network of infected hosts would be collisions. This was found to be the case for the 151.62.222.0/24 network, where 164 collisions took place. This high collision rate (note that the other 92 non-collisions in the network would almost certainly be due to hosts that were not configured to be vulnerable to the exploits used in the worms) confirms that the simulation operates as expected.

6.7.3 Network Telescope

The final simulation concerns a form of Worm countermeasure testing. Network telescopes, discussed in Chapter 2, can be used to find infected hosts (or hosts with malicious intent) and block them from sending traffic to a group of other hosts using "Real-Time Blackhole Lists".

This simulation tests the effectiveness of this technology as a means of limiting the amount of damage a worm can cause.

Statement of Subject

The focus of this simulation is the network telescope component. It operates in the following way: if any packets are routed to a specific network (in this case, the 146.231.0.0/16 Class B network) then the sending host is recorded. Another set of networks (in this case, the 196.115.0.0/16 Class B network) "subscribes" to the telescope's list (or RBL, "Real-time Black-hole List"), and will not be infected should they receive packets from the hosts on the telescope's list.

The test associated with this simulation is to determine the usefulness of network telescopes as tools for worm deterrence. If the telescope causes a notable difference in worm scan patterns, then it will be deemed as an effective tool.

Statement of Parameters

A single host on a small simulated network of 16 hosts is infected with a perfectly infectious random-scanning worm. The networks specified above will be created with the special cases they require (network telescope with RBL and RBL subscriber), and the worm will be executed for one million ticks (corresponding to approximately 990 000 infection cycles). This is reduced from the default due to the rapid propagation and perfect infection of the worm.

Statement of Results

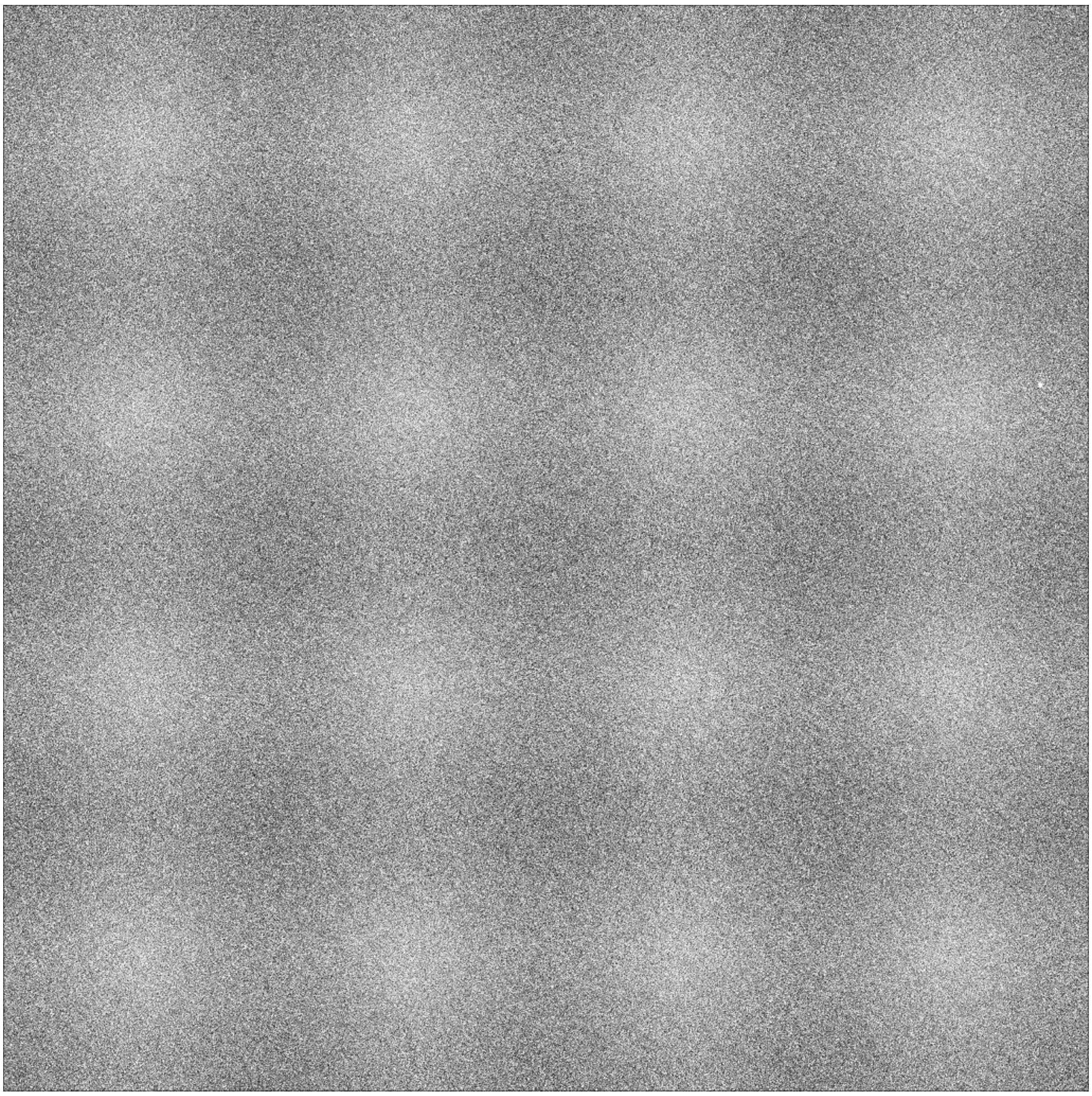

Figure 6.14: Network Telescope Hilbert Curve

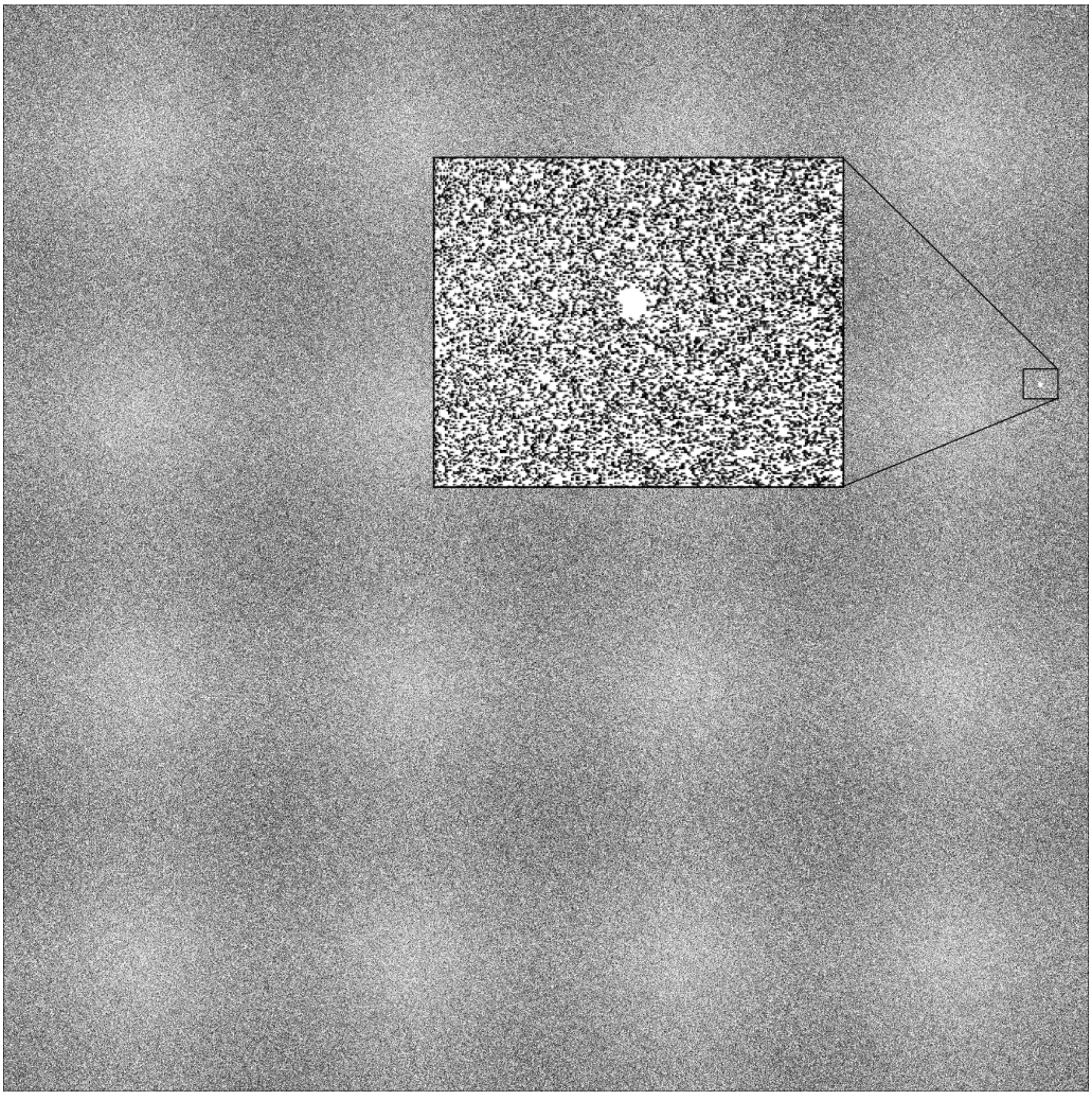
Figure 6.15: Magnified Network Telescope Hilbert Curve

The final result shows a similar “snow” or “static” effect as seen in the Simple worm simulation (Section 6.6.1) when visualised in Figure 6.14. Throughout the course of the simulation, the network telescope was observed for usefulness, and it was found to be effective - approximately one hundred and fifty infection packets over the course of the full simulation were rejected as a result of the RBL.

While initially it appears that the scanning pattern shows no effect, the protected network can clearly be seen when magnified, as shown in Figure 6.15.

This shows that the network telescope component did operate successfully, and effectively stopped the randomly scanning worms from infecting their “protected” network. If the worm had used an ordered infection pattern (such as those used by the Blaster or Welchia worms) and the telescope was scanned first, then it would have been more effective as a denser clustering of scans on the specific networks (telescope and subscriber) could be expected. However, if the worm had first infected the protected network, it would have had no effect at all.

Commentary

The network telescope simulation used a simulated network telescope to act as a worm countermeasure. Within 10 000 ticks, all infectable hosts on the local network (16 in total) had been infected, and within 50 000 ticks all had been detected by the telescope. As a result, no packets were detected in the 196.115.0.0/16 network - where other networks typically experienced a range of 'hits' between 130 and 180 packets, the 196.115.0.0/16 network received no packets before the telescope had detected the infecting host and added it to the RBL.

This is not what was expected - it is significantly more effective than originally thought! The simulator did register approximately 150 hits that were blocked due to the RBL, which is the expected response in the expected quantities. This does confirm the effectiveness of the telescope, as well as the simulator's capability of modeling it.

6.7.4 Advanced Simulations Results

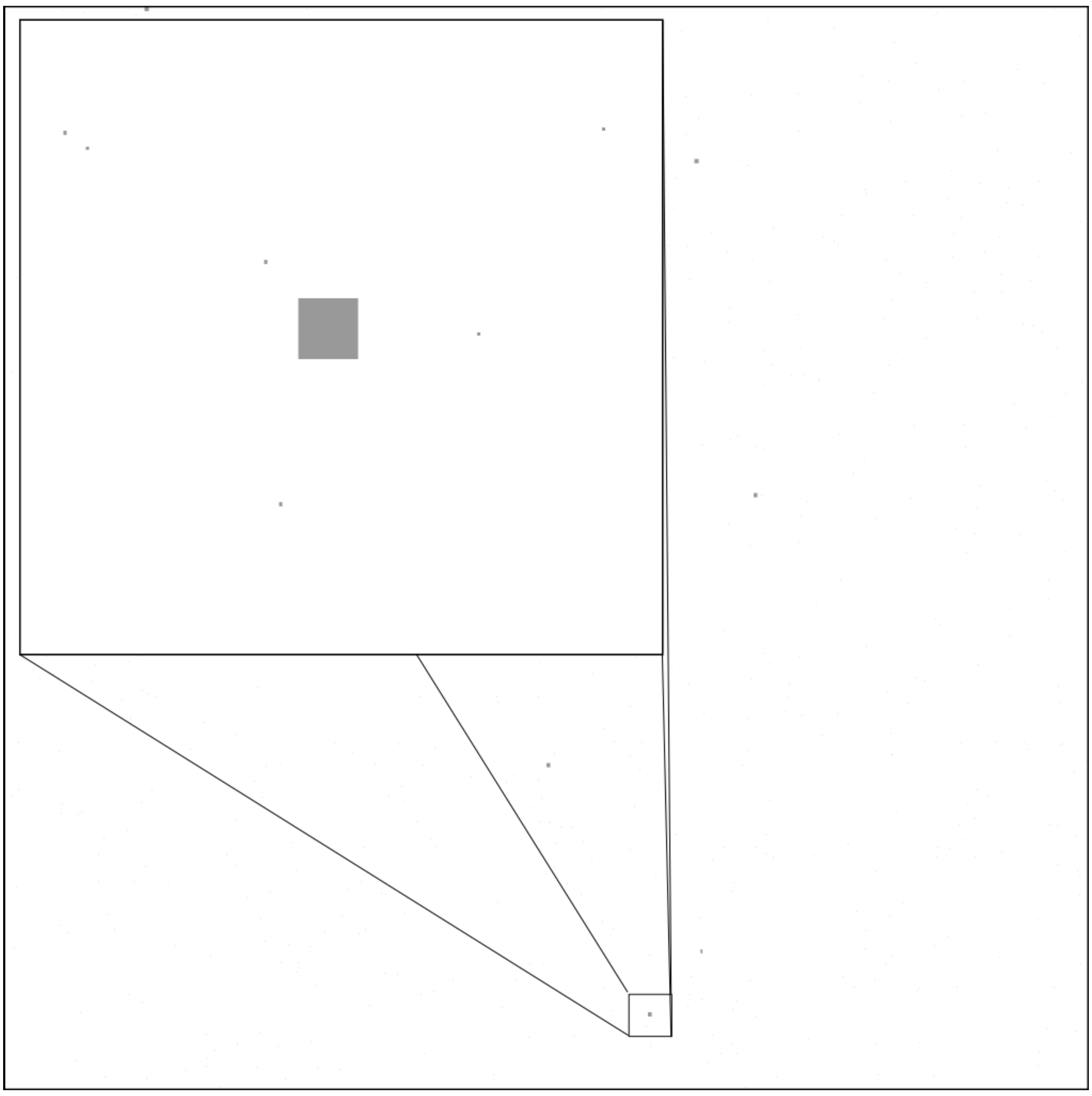

Figure 6.16: Red vs. Blue worm with Magnification

The advanced simulations (Red vs. Blue worm and network telescope simulations) both showed that the proposed worm countermeasures, while effective if targeted, were rarely useful due to the large size of IP space. In Figure 6.16 it is quite clearly possible to see the Class B network that the Welchia worm infected, with nearby Class C networks infected by Blaster. Throughout the entire course of the simulation, no collision took place.

This does not mean that these worm defence tools were ineffective - it rather means that they did not have an effect on an Internet scale. They provided excellent security for specific networks on the Internet, but were limited due to their lack of ubiquity.

6.8 Other Simulators

In Section 2.6, two other major simulators were mentioned: ns-2 (as a simulator) and SSFNet (as a simulator library). A further simulator was considered for comparison in this research: the commercial network simulator OPNET (OPNET Technologies (2009)). Due to restricted access, it was rejected as a possible simulator for comparison.

The other simulators (ns-2 and SSFNet) were available for comparison in this research, but were not used due to resource and time constraints. The peculiarities of their use (ns-2's use of dual-languages and emulation focus, and SSFNet's development requirements) meant that this research could not be easily recreated for their frameworks.

In addition to this, the indices used for simulator testing were not comparable. The simulations that were executed were heavily investigated for accuracy to expected results - and were found to meet these expectations. There is no reason to assume that ns-2 or SSFNet would have yielded any different results.

6.9 Summary

To conclude the series of tests prepared and executed on the developed simulation framework and engine in Chapter 5, we can see that it successfully executed a wide variety of simulated networking examples spanning normal networking (such as the routing and protocol simulations) as well as malware-specific simulations (such as the various worm scanning algorithms).

The results discussed in this chapter show that the simulator meets the goals set out for it, namely it has been stress-tested using Internet worms, and has been found to be capable of simulating large models while still achieving the expected outcomes for each test.

The next chapter concludes this work, commenting on the results achieved in this and the previous chapter, considering the achievements, shortcomings, and possible extensions to this work.

7 Conclusion

7.1 Introduction

In this final chapter, the conclusions regarding this work are drawn and analysed. In Chapters 5 and 6, the simulations around which this work revolves were executed - in Section 5.5 these results are discussed.

Section 7.4 covers the potential extensions to this research and the deliverable simulator that it has created, while Section 7.5 suggests the applications that this work has in academia and research, contrasting the benefits against the stated shortcomings in Section 7.6.

Finally, Section 7.7 draws the final summary of this research together, concluding this work.

7.2 Goal Review

The goals stated in Section 1.4 bear reflection upon completion of this research. The two goals, as stated, were the development of a set of recommendations for constructing robust Internet-scale simulators, and the development of a simulator using those recommendations. They are reviewed and considered here.

7.2.1 Recommendations

Chapters 3 and 4 related a variety of experiences that a developer of a generic network simulator should experience, as well as documenting how the challenges were overcome.

Chapter 3 states the software recommendations of developing the simulator, while Chapter 4 discusses the specifics of how the simulator was constructed.

The set of recommendations proved to be sufficient for the construction of the simulator, detailed in the next section. The recommendations covered both the theoretical (design) and the practical (implementation) aspects of simulator design, on a broad level as well as including many practical observations.

As such, the development of recommendations for the construction of these simulators can be considered successful. They can be used by other researchers in this subject area as a basis for their own simulator development.

7.2.2 Simulator

The simulator that was to be developed as a goal for this research was completed early in the research and then continually improved, as explained in Chapter 3. It has evolved through iterative development cycles to allow for easier development and more efficient access to resources.

The evidence of the success of the simulator can be found in Chapters 5 and 6. The results of Chapter 5 show that the simulator is capable of a broad variety of aspects of networking and the results of Chapter 6 show that the simulations it performs are capable of focused research into a particular domain.

Specifically, the simulator succeeded in its chief goal - it is both robust, capable of a diverse range of simulation, and large in scale, capable of simulating millions of events across the full IPv4 Internet. It has been shown to be scalable, via the distributed memory prototype, but that component of the system must still be matured before becoming a core part of the simulator.

The simulator has shown itself to be capable of Internet malware simulation, achieving every expected outcome presented in this document.

This goal, then, can also be considered successfully completed.

7.3 Reflections

Chapter 2 reviewed some of the literature available on network simulations, Internet worms and network simulators. This research made use of these resources to yield the specifications for creating a network simulator, detailed in Chapters 3 and 4.

It then proposed a series of simulations that would test the operations of a simulator developed according to the specifications proposed. The results can be found in Chapters 5 and 6.

While the simulator presented has adequately completed all of the requirements set before it (as shown above in Section 7.2), it does not have the resources at the disposal of the larger network simulators (such as ns-2 or SSFNet, detailed in Section 2.6). It offers features that neither of these simulators have reproduced: using a highly cross-platform and language-agnostic approach, it allows for scalable network simulations - but it does not have the large component/plugin sets that these simulators can boast.

It has been effective in the rapid development of plugins and simulations, and has been shown to be extensible via the grid prototyping, discussed in Section 4.4. It has also been shown as effective in running large simulations, representing millions of hosts (albeit via abstraction using network nodes) over the course of tens of millions of ticks. These simulations have accurately depicted the expected results - the results of Internet worms have been compared to the expected results and found to match closely.

Section 7.2 compares the expectations of the research to the achievements, and finds that the research in this work can be considered a success.

7.4 Extensions

Due to the modular nature of the simulator, extensions in terms of simulations are very simple to suggest: any form of network research can be investigated using the simulator to test hypotheses. However, the challenge in further development of the simulation engine itself is also imperative.

Simulations which logically extend the research in Internet worm simulation that could be used might include:

- The development of cutting-edge countermeasures simulations added to the system for testing and evaluation (such as increasing the focus on tarpits and RBLs and network telescopes)
- Testing various novel worm concepts and existing concepts which were out of the scope of this document (such as crypt-virii, flashworms, and bot-nets)
- Using existing worm datasets to test and calibrate the accuracy of the simulator against real-world data. Captures of real world worm traffic are available from organisations such as CAIDA (see Cooperative Association for Internet Data Analysis (2008b)), however there are limitations in accessing these given the scarcity of connectivity in South Africa, and the datasets are very large.

Extensions to the simulator that could result in further extensions to the work include the following:

- The simulator has been designed to operate upon a grid (which is part of the design concept), but was never properly implemented beyond the prototype discussed in Section 4.4 due to time and technical constraints. By adding additional grid capability to the simulator (by offloading additional processing and optimising grid memory distribution), further grid development will increase the memory capacity and processing power available to simulations.
- The initial concept for the simulator included a GUI design. This was later discarded as a secondary priority, but would add a layer of user-friendliness and usability that would make the system more useful to non-developers.

- Constructing more diverse plugins for packaging with the simulator will make simulations even easier to run for researchers. The existing plugins are sufficient for worm research and basic IPv4 testing, but lower and higher level protocol plugins would make other research simpler.

Finally, an additional avenue of research, which Section 6.8 shows was not considered in this document, was to compare efficiency and scalability of this simulator with those of other simulators.

These extensions are all within the scope of a capable developer or researcher, while providing valuable additions to the network simulator described in this document.

7.5 Applications of This Work

This document and the simulator that it describes can provide a valuable set of guidelines to network and security professionals and researchers.

The document details the construction of a tool which security or research institutes could implement for themselves, with optimisations and additional details added according to their needs. The importance of the robustness of their simulations has been discussed, and it is hoped that researchers will take the advice into consideration: robust simulators allows for simple extensions to be rapidly developed outside of the bounds of simpler, domain-specific simulators which are commonly found in academia.

The simulator that is used in this work is also highly applicable to security researchers, particularly in the area of worm research. Commonly used plugins have already been developed, and due to the simulator's cross-platform nature it is very easy for other researchers in the field to run their simulations, with little thought on underlying architecture required. It has many example simulations already available, and it is open-source in order to allow others to learn from the existing work.

Finally, this document itself brings together much research in the fields of security and simulation, and shows the importance of simulation as an important research tool in this field.

7.6 Shortcomings

This reseach, while valuable, has several shortcomings that must be acknowledged.

7.6.1 Efficiency

Firstly, while efficiency of simulation has been maintained, it came at the cost of accuracy (due to the loss of granularity). This is an expensive trade-off, which could be reduced by more efficient coding and data structures (particularly, in a more efficient memory management system). The grid prototype discussed in Section 4.4 is one way in which efficiency has been addressed - further development of distributed processing is an immediate step towards resolving this shortcoming.

7.6.2 Development

The simulator currently requires .NET or mono development for simulations, even those using the simplest available plugins. As stated in Section 7.4, the initial concept for the simulator included a GUI in order to allow non-developers to use the simulator (despite being limited by the inability to create new plugins).

7.6.3 IPv4 Limitation

The simulator is currently set to only use IPv4 addresses for hosts. This removes several simulation options (such as OSI layer 1 and 2 simulations). This limitation is artificial, as any research into non-IPv4 networking can easily add such capability, but the networking at present is not perfectly robust.

7.6.4 Time Scale

Because the simulator operates in abstract ‘ticks’, which bear no resemblance to any real time period, it is challenging to associate a specific time in a simulation with time in the real world. When testing, this is artifically avoided by scaling time to match known infection spread times, but when performing speculative simulations the results may differ significantly.

7.7 Final Summary

In conclusion, it is necessary to restate the importance of simulation in security research. As the broad body of research literature shows, simulation is an imperative part of the testing and development process, especially in a field like security where research can involve sensitive information which should not be exposed to non-researchers, elements which are dangerous if directly reproduced, or controlled environments for testing which are difficult to create.

The development of simulators which can execute complex simulations which can be easily extended is preferable to “one-shot” simulations, as research benefits from tools that can be altered to fit the needs of the problems considered. Futhermore, by constructing plugin architectures that encourage modularity, creation of complex network simulations is possible with little effort if the required plugins exist or are similar to existing plugins. Another aspect that is important to these simulators is efficiency (to the extent that they can execute a variety of simulations on a researcher's desktop PC).

The best way to test all of the above attributes of the system is via a load-heavy series of simulations. An example of this is Internet worm simulation, as worms grow quickly and affect a diverse range of networking components, as well as being well documented.

A simulator was developed for this research, and showed promising results. A selection of tests in the form of simulations was executed upon the simulator, and the results show that it does succeeds in simulating known worm behaviour, meeting the expected outcomes for every simulation.

References

F. Baccelli and D. Hong. Flow level simulation of large IP networks. In *IEEE Conference on Computer Communications 2003*, volume 3, pages 1911–1921, 2003.

Y. Bai, A. T. Ogielski, and G. Wu. Interactions of TCP and radio link ARQ protocol. In *Vehicular Technology Conference, 1999*, volume 3, pages 1710–1714, 1999. doi: 10.1109/VETECF.1999.801596.

M. Bailey, E. Cooke, F. Jahanian, D. Watson, and J. Nazario. The Blaster Worm: Then and Now. *Security and Privacy, IEEE*, 3:26–31, 2005.

A. L. Barabasi and R. Albert. Emergence of Scaling in Random Networks. In *Science*, volume 286, pages 509–512, October 1999.

L. Briesemeister and P. Porras. Microscopic simulation of a group defense strategy. In *PADS '05: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, pages 254–261, Washington, 2005. IEEE Computer Society. ISBN 0-7695-2383-8. doi: http://dx.doi.org/10.1109/PADS.2005.13.

T. Bu and D. Towsley. On distinguishing between internet power law topology generators. In *Proceedings of the IEEE Conference on Computer Communications*, 2002.

M. Cai, K. Hwang, J. Pan, and C. Papadopoulos. WormShield: Fast Worm Signature Generation with Distributed Fingerprint Aggregation. *IEEE Transactions on Dependable and Secure Computing*, 4(2):88–104, 2007. ISSN 1545-5971. doi: http://doi.ieeecomputersociety.org/10.1109/TDSC.2007.1000.

F. Castaneda, E. C. Sezer, and J. Xu. Worm vs. worm: preliminary study of an active counter-attack mechanism. In *WORM '04: Proceedings of the 2004 ACM Workshop on Rapid malcode*, pages 83–93, New York, 2004. ACM. ISBN 1-58113-970-5. doi: http://doi.acm.org/10.1145/1029618.1029631.

B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. *Wireless Networking*, 8(5):481–494, 2002a. ISSN 1022-0038. doi: http://dx.doi.org/10.1023/A:1016542229220.

Z. Chen, L. Gao, and K. Kwiat. Modeling the spread of active worms. *IEEE Conference on Computer Communications 2003*, 3, 2002b.

Computer Emergency Response Team. CERT advisory CA-1992-02 michelangelo pc virus warning. [Website, Accessed: 27/11/2008], 1992. URL `http://www.cert.org/advisories/CA-1992-02.html`.

Cooperative Association for Internet Data Analysis. CAIDA Network Telescope Data. [Website, Accessed: 20/6/2008], 2008a. URL `http://www.caida.org/research/security/telescope/`.

Cooperative Association for Internet Data Analysis. Ucsd network telescope – code-red worms dataset. [Website, Accessed 21/11/2008], 2008b. URL `http://www.caida.org/data/passive/codered_worms_dataset.xml`.

S. E. Coull and B. K. Szymanski. On the development of an internetwork-centric defense for scanning worms. In *HICSS '07: Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, page 144a, Washington, 2007. IEEE Computer Society. ISBN 0-7695-2755-8. doi: http://dx.doi.org/10.1109/HICSS.2007.406.

J. Cowie and H. Liu. Towards Realistic Million-Node Internet Simulations. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications*, 1999. URL `http://www.ssfnet.org/Papers/pdpta99.pdf`.

J. H. Cowie, D. M. Nicol, and A. T. Ogielski. Modeling the global Internet. *Computing in Science and Engineering*, 1(1):42–50, January 1999. URL `http://citeseer.ist.psu.edu/cowie99modeling.html`.

R. Danyliw and A. Householder. CERT advisory CA-2001-19 "Code Red" Worm Exploiting Buffer Overflow in IIS indexing service DLL. [Website, Accessed: 27/11/2008], 2001. URL `http://www.cert.org/advisories/CA-2001-19.html`.

C. Dougherty, J. Havrilla, S. Hernan, and M. Lindner. CERT advisory CA-2003-20 W32/Blaster worm. [Website, Accessed: 27/11/2008], 2003. URL `http://www.cert.org/advisories/CA-2003-20.html`.

D. Dutta, A. Goel, and J. Heidemann. Faster network design with scenario prefiltering. In *Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 237–246, Fort Worth, October 2002. USC/Information Sciences Institute, IEEE. URL `http://www.isi.edu/~johnh/PAPERS/Dutta02d.html`.

T. D. Dyer and R. V. Boppana. A comparison of TCP performance over three routing protocols for mobile ad hoc networks. In *MobiHoc '01: Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 56–66, New York, 2001. ACM. ISBN 1-58113-428-2.

F-Secure Corporation. F-secure virus descriptions : Stoned. [Website, Accessed: 27/11/2008], 2008. URL `http://www.f-secure.com/v-descs/stoned.shtml`.

K. Fall and K. Varadhan. The ns manual. Technical report, The VINT Project, December 2008. URL `http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf`.

J. Farber, S. Bodamer, and J. Charzinski. Measurement and modelling of internet traffic at access networks, 1998. URL `http://citeseer.ist.psu.edu/441125.html`.

A. Feldmann, A. C. Gilbert, P. Huang, and W. Willinger. Dynamics of IP traffic: a study of the role of variability and the impact of control. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 301–313, New York, 1999. ACM. ISBN 1-58113-135-6. doi: http://doi.acm.org/10.1145/316188.316235.

S. Friedl. Analysis of the new "Code Red II" variant. [Website, Accessed 20/6/2008], August 2001. URL `http://www.unixwiz.net/techtips/CodeRedII.html`.

M. Garetto, R. L. Cigno, M. Meo, and M. Marsan. A detailed and accurate closed queueing network model of many interacting TCP flows. In *IEEE Conference on Computer Communications 2001*, 2001.

A. Gatti, Milidonis, and AGold. Internet security systems PAM ICQ server response processing vulnerability. [Website, Accessed 3/12/2008], 2004. URL `http://research.eeye.com/html/advisories/published/AD20040318.html`.

S. P. Gorman, R. G. Kulkarni, L. A. Schintler, and R. R. Stough. A predator prey approach to the network structure of cyberspace. In *WISICT '04: Proceedings of the winter international synposium on Information and communication technologies*, pages 1–6. Trinity College Dublin, 2004.

C. Hanle and M. Hofmann. Performance comparison of reliable multicast protocols using the network simulator ns-2. In *23rd Annual IEEE International Conference on Local Computer Networks*, page 222, 1998.

T. Henderson, E. Sahouria, S. McCanne, and R. Katz. On improving the fairness of TCP congestion avoidance. In *Global Telecommunications Conference, 1998*, volume 1, 1998. doi: 10.1109/GLOCOM.1998.775786.

M. Hofmann, E. Ng, K. Guo, S. Paul, and H. Zhang. Caching techniques for streaming multimedia over the internet. Technical report, Bell Laboratories, April 1999. URL `http://citeseer.ist.psu.edu/hofmann00caching.html`.

Y. C. Hu and D. B. Johnson. Caching strategies in on-demand routing protocols for wireless ad hoc networks. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 231–242, New York, 2000. ACM. ISBN 1-58113-197-6. doi: http://doi.acm.org/10.1145/345910.345952.

P. Huang, D. Estrin, and J. Heidemann. Enabling Large-scale Simulations: Selective Abstraction Approach to The Study of Multicast Protocols. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 241–248, 1998. doi: 10.1109/MASCOT.1998.693701.

I. A. N. A. (IANA). RFC 3330: Special-use ipv4 addresses. Technical report, Internet Engineering Task Force, September 2002. URL `http://www.ietf.org/rfc/rfc3330.txt`.

Information Sciences Institute. RFC 760: Internet protocol. Technical report, University of Southern California, January 1980a. URL `http://www.ietf.org/rfc/rfc760.txt`.

Information Sciences Institute. RFC 761: Transmission control protocol. Technical report, University of Southern California, January 1980b. URL `http://www.ietf.org/rfc/rfc761.txt`.

International Telecommunication Union. Data networks and open system communications: Open systems interconnection - model and notation (X.200). Technical report, Telecommunication and Standardization Sector of ITU, 1994.

Iowa State University Information Assurance Center. ISEAGE: Internet-Scale Event and Attack Generation Evironment. [Website, Accessed 21/11/2008], 2008. URL `http://www.iac.iastate.edu/iseage/`.

B. Irwin and N. Pilkington. High level internet level traffic visualization using hilbert curve mapping. In J. R. Goodall, G. Conti, and K.-L. Ma, editors, *VizSEC 2007: Proceedings of the Workshop on Visualization for Computer Security*, Mathematics and Visualisation, pages 147–158. Springer Berlin Heidelberg, 2008. URL `http://www.springerlink.com/content/p3642870865k1621/`.

V. Jacobson. Congestion avoidance and control. *SIGCOMM Computing Communication Review*, 25(1):157–187, 1995. ISSN 0146-4833. doi: http://doi.acm.org/10.1145/205447.205462.

J. L. Jeremie. Analysis of the internet topology, 2004. URL http://citeseer.ist.psu.edu/748554.html.

Y. Joo, V. Ribeiro, A. Feldmann, A. C. Gilbert, and W. Willinger. On the impact of variability on the buffer dynamics in IP networks. In *37th Annual Allerton Conference on Communication, Control, and Computing*. Alerton, IL, 1999. URL http://www.dsp.rice.edu/publications.

V. Kanodia, C. Li, A. Sabharwal, B. Sadeghi, and E. Knightly. Distributed multi-hop scheduling and medium access with delay and throughput constraints. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 200–209, New York, 2001. ACM. ISBN 1-58113-422-3. doi: http://doi.acm.org/10.1145/381677.381697.

F. Kelly. Mathematical modelling of the internet. In B. Engquist and W. Schmid, editors, *Mathematics Unlimited – 2001 and Beyond*, pages 685–702. Springer-Verlag, Berlin, 2001. URL http://citeseer.ist.psu.edu/article/kelly99mathematical.html.

W. O. Kermack and A. G. McKendrick. A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society of London*, 115:700–721, 1927.

H. A. Kim and B. Karp. Autograph: toward automated, distributed worm signature detection. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 19–19, Berkeley, 2004. USENIX Association.

D. Knowles and F. Perriott. W32.blaster.worm. [Website, Accessed 20/6/2008], August 2003. URL http://www.symantec.com/security_response/writeup.jsp?docid=2003-081113-0229-99&tabid=2.

J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang. Providing robust and ubiquitous security support for mobile ad-hoc networks. In *International Conference on Network Protocols (ICNP)*, pages 251–260, 2001. URL http://citeseer.ist.psu.edu/article/kong01providing.html.

LaBrea Tarpits. Labrea: "sticky" honeypot and IDS. [Website, Accessed 10/12/2008], December 2008. URL http://labrea.sourceforge.net.

K. Lan and J. Heidemann. A tool for rapid model parameterization and its applications. In *MoMeTools '03: Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pages 76–86, New York, 2003. ACM Press. ISBN 1-58113-748-8. doi: http://doi.acm.org/10.1145/944773.944786.

B. M. Leiner, V. G. Cerf, D. D. Clark, R. E. Kahn, L. Kleinrock, D. C. Lynch, J. Postel, L. G. Roberts, and S. Wolf. A brief history of the internet, 1999. URL `http://www.citebase.org/abstract?id=oai:arXiv.org:cs/9901011`.

K. Li, C. Pu, and M. Ahamad. Resisting spam delivery by TCP damping. In *First Conference on Email and Anti-Spam (CEAS)*, July 2004. URL `http://citeseer.ist.psu.edu/li04resisting.html`.

L. Li. The speedup of SSFNet simulation over NS. Master's thesis, Rensselaer Polytechnic Institute, 2001.

J. Licklider and W. Clark. On-Line Man Computer Communication. Memos, August 1962.

M. Liljenstam, D. Nicol, B. Premore, and Y. Yuan. A mixed abstraction level simulation model of large-scale internet. In *MASCOTS 2002. 10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*, 2002.

M. Liljenstam, D. M. Nicol, V. H. Berk, and R. S. Gray. Simulating realistic network worm traffic for worm warning system design and testing. In *WORM '03: Proceedings of the 2003 ACM workshop on Rapid malcode*, pages 24–33, New York, 2003. ACM Press. ISBN 1-58113-785-0. doi: http://doi.acm.org/10.1145/948187.948193.

Z. M. Mao, R. Govindan, G. Varghese, and R. H. Katz. Route flap damping exacerbates internet routing convergence. *SIGCOMM Computing Communication Review*, 32(4): 221–233, 2002. ISSN 0146-4833. doi: http://doi.acm.org/10.1145/964725.633047.

M. Marina and S. R. Das. On-demand multipath distance vector routing in ad hoc networks. In *Network Protocols Ninth International Conference on ICNP 2001*, pages 14– 23, 2001.

Microsoft. Microsoft Security Bulletin MS03-026: Buffer Overrun In RPC Interface Could Allow Code Execution (823980). [Website, Accessed 10/11/2008], September 2003a. URL `http://www.microsoft.com/technet/security/bulletin/MS03-026.mspx`.

Microsoft. Microsoft Security Bulletin MS03-039: Buffer Overrun In RPCSS Service Could Allow Code Execution (824146). [Website, Accessed 10/11/2008], September 2003b. URL `http://www.microsoft.com/technet/security/bulletin/MS03-039.mspx`.

Microsoft. Microsoft Security Bulletin MS01-033: Unchecked Buffer in Index Server ISAPI Extension Could Enable Web Server Compromise. [Website, Accessed 10/11/2008], November 2003c. URL `http://www.microsoft.com/technet/security/bulletin/MS01-033.mspx`.

D. Moore. Network Telescopes: Observing Small or Distant Security Events. Slideshow. [Website, Accessed: 20/6/2008], August 2002.

D. Moore, C. Shannon, G. M. Voelker, and S. Savage. Network telescopes. Technical report, Cooperative Association for Internet Data Analysis, San Diego Supercomputer Center, University of California, San Diego and Department of Computer Science and Engineering, University of California, San Diego, 2004.

J. Nazario. *Defense and Detection Strategies against Internet Worms.* Artech House Publishers, October 2003. ISBN 1580535372.

Net Applications. Operating system market share. [Website, Accessed 26/11/2008], November 2008. URL `http://marketshare.hitslink.com/report.aspx?qprid=8`.

D. M. Nicol. Fluid simulation: discrete event fluid modeling of TCP. In *Proceedings of the 33nd conference on Winter simulation*, Arlington, December 2001.

OPNET Technologies. Opnet technologies website. [Website, Accessed 13/3/2009], March 2009.

Oxford English Dictionary Online. [Website, Accessed: 18/6/2008], June 2008. URL `http://www.oed.com`.

R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of internet background radiation. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 27–40, New York, 2004. ACM. ISBN 1-58113-821-0. doi: http://doi.acm.org/10.1145/1028788.1028794.

V. Paxson and S. Floyd. Why we don't know how to simulate the internet. In *29th conference on Winter simulation*, pages 1037 – 1044, Atlanta, 1997.

F. Perriot. Symantec W32.Welchia.Worm information. [Website, Accessed 27/11/2008], 2008. URL `http://www.symantec.com/security_response/writeup.jsp?docid=2003-081815-2308-99`.

L. F. Perrone and D. M. Nicol. A scalable simulator for TinyOS applications. In *Simulation Conference, 2002. Proceedings of the Winter*, volume 1, pages 679– 687, December 2002. doi: 10.1109/WSC.2002.1172947.

R. Puri, K. W. Lee, K. Ramchandran, and V. Bharghavan. An integrated source transcoding and congestion control paradigm for video streaming in the internet. In *IEEE Transactions on Multimedia*, volume 3, March 2001.

D. Rao and P. A. Wilsey. Multi-resolution network simulations using dynamic component substitution. In *Proceedings of the Ninth International Symposium in Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01) table of contents*, page 142, Washington, 2001. IEEE Computer Society. URL `http://citeseer.ist.psu.edu/629417.html`.

R. Rejaie, M. Handley, and D. Estrin. RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the internet. In *Proceedings of IEEE INFO-COM*, volume 3, pages 1337–1345, March 1999.

Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. RFC 1918 - address allocation for private internets. Technical report, Internet Engineering Task Force, February 1996. URL `http://www.ietf.org/rfc/rfc1918.txt`.

J. Richter and B. Irwin. Analyzing the noise of the internet sampled using a small south african network telescope. [Unpublished], November 2008.

S. Sahu, P. Nain, C. Diot, V. Firoiu, and D. Towsley. On achievable service differentiation with token bucket marking for TCP. In *SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 23–33, New York, 2000. ACM. ISBN 1-58113-194-1. doi: http://doi.acm.org/10.1145/339331.339342.

K. Salamatian and S. Vaton. Hidden Markov modeling for network communication channels. In *SIGMETRICS/Performance*, pages 92–101, 2001.

S. Savage, T. Anderson, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: The internet's scalability. *IEEE Micro*, 19:50–59, 1999. URL `http://citeseer.ist.psu.edu/737245.html`.

C. Shannon and D. Moore. The Spread of the Witty Worm. [Website, Accessed 10/11/2008], February 2007. URL `http://www.caida.org/research/security/witty/`.

M. I. Sharif, G. F. Riley, and W. Lee. Comparative study between analytical models and packet-level worm simulations. In *19th Workshop on Principles of Advanced and Distributed Simulation*, pages 88–98, Los Alamitos, 2005. IEEE Computer Society. doi: http://doi.ieeecomputersociety.org/10.1109/PADS.2005.5.

P. Sinha, R. Sivakumar, and V. Bharghavan. Enhancing ad hoc routing with dynamic virtual infrastructures. In *Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1763–1772, 2001.

J. Stewart. Witty Worm Analysis. [Website, Accessed 10/11/2008], March 2004. URL `http://www.secureworks.com/research/threats/witty/`.

A. Veres, K. S. Molnár, and G. Vattay. On the propagation of long-range dependence in the internet. *SIGCOMM Computing Communication Review*, 30(4):243–254, 2000. ISSN 0146-4833. doi: http://doi.acm.org/10.1145/347057.347551.

A. Wagner, T. Dubendorfer, B. Plattner, and R. Hiestand. Experiences with worm propagation simulations. In *WORM '03: Proceedings of the 2003 ACM workshop on Rapid malcode*, pages 34–41, New York, 2003. ACM Press. ISBN 1-58113-785-0. doi: http://doi.acm.org/10.1145/948187.948194.

K. Wang, G. Cretu, and S. J. Stolfo. Anomalous payload-based worm detection and signature generation. In *Lecture Notes in Computer Science*, volume 3858/2006, pages 227–246. Springer Berlin/Heidelberg, 2006. doi: 10.1007/11663812.

N. Weaver, I. Hamadeh, G. Kesidis, and V. Paxson. Preliminary results using scale-down to explore worm dynamics. In *WORM '04: Proceedings of the 2004 ACM workshop on Rapid malcode*, pages 65–72, New York, 2004. ACM Press. ISBN 1-58113-970-5. doi: http://doi.acm.org/10.1145/1029618.1029628.

M. M. Williamson and M. M. Williamson. Design, implementation and test of an email virus throttle. In *In Proceedings of the 19th Annual Computer Security Applications Conference, Las Vegas*, 2003.

J. Winick and S. Jamin. Inet-3.0: Internet topology generator. Technical report, University of Michigan, 2002.

N. R. Wyler, N. Archibald, S. Fogie, C. Hurley, D. Kaminsky, J. Long, N. Marigoni, L. McOmie, H. Meer, B. Potter, and R. Temmingh. *Aggressive network self-defense.* Syngress media, Rockland, 2005. ISBN 1-931836-20-5. URL `http://site.ebrary.com/lib/ucsc/Doc?id=10075669(Webview)`.

Y. Xiang and W. Zhou. Protect grids from DDoS attacks. In *3rd International Conference on Grid and Cooperative Computing*, pages 309–317, 2004.

Y. Xu, J. Heidemann, and D. Estrin. Adaptive energy-conserving routing for multihop ad hoc networks. Research Report 527, USC/Information Sciences Institute, October 2000. URL `http://www.isi.edu/~johnh/PAPERS/Xu00a.html`.

V. Yegneswaran, P. Barford, and D. Plonka. The design and use of internet sinks for network abuse monitoring. In *Recent Advances in Intrusion Detection*, 2004. URL `http://citeseer.ist.psu.edu/yegneswaran04design.html`.

R. H. Zakon. Hobbes' internet timeline v8.2. [Website, Accessed 09/12/2008], November 2006. URL `http://www.opnet.com`.

Y. K. Zhang, F. W. Wang, Y. Q. Zhang, and J. F. Ma. Worm propagation modeling and analysis based on quarantine. In *InfoSecu '04: Proceedings of the 3rd International Conference on Information Security*, pages 69–75, New York, 2004. ACM Press. ISBN 1-58113-955-1. doi: http://doi.acm.org/10.1145/1046290.1046305.

S. Zhou and R. J. Mondragon. Towards modelling the internet topology - the interactive growth model. In *18th International Teletraffic Congress*, volume 5a, page 121, 2003.

C. C. Zou and W. Gong. Email worm modeling and defense. *Conference on Computer Communications and Networks*, pages 409–414, 2004.

C. C. Zou, W. Gong, and D. Towsley. Code red worm propagation modeling and analysis. In *CCS '02: Proceedings of the 9th ACM conference on Computer and Communications Security*, pages 138–147, New York, 2002. ACM Press. ISBN 1-58113-612-9. doi: http://doi.acm.org/10.1145/586110.586130.

C. C. Zou, W. Gong, and D. Towsley. Worm propagation modeling and analysis under dynamic quarantine defense. In *WORM '03: Proceedings of the 2003 ACM workshop on Rapid Malcode*, pages 51–60, New York, 2003. ACM Press. ISBN 1-58113-785-0. doi: http://doi.acm.org/10.1145/948187.948197.

Appendix A: Glossary

Connection A component of a computer network which models any form of communication between two nodes.

Hilbert Curve A fractal space-filling curve that allows one dimensional data to be represented in multiple dimensions, while maintaining locality - effective in representing IP addresses in a square visual while the position of the data retains significant semantic value.

Host Any node to which information can be addressed.

Internet Worm Malicious software that can propagate without motivation - it requires tangible interaction with the Internet in order to spread.

IP Internet Protocol, a means of networked communication which uses four octets for addressing, the basis for the modern Internet.

Malware Malicious software, executable information which can execute code that the would be detrimental to the system upon which it resides.

Node Any component of a computer network which can transfer information.

OSI The Open Systems Interconnect model of networking explains protocols using a "stack". Each layer of the stack is interchangable, with lower layers representing the communication and physical protocols while high layers represent the application-specific protocols.

Packet In simulated networking, a component that may contain information. In real networking, a conceptual unit of data transferral which can be part of a larger "stream" of data, or an isolated single communique.

Simulations A single modeled scenario in which aspects of the real world are depicted, typically with changes depicted over time. Useful for speculation, plan testing and prototyping.

Simulator A piece of software that executes simulations, including storing the simulated components in memory, executing changes upon those components, and presenting these changes to the user.

TCP Transmission Control Protocol, a protocol that abstracts “streams” of information by using IP as an underlying communication protocol while adding features to ensure that data is communicated without loss of integrity.

UDP A connectionless alternative protocol to TCP, it sends packets without the necessity of an abstract “connection”. This results in lower transmission overhead, but no guarantee of successful transmission.

Appendix B: Example Results

An example of output from the simulator developed in Chapters 3 and 4 and used in Chapters 5 and 6 is presented here.

This example output is from a simulation similar to the Blaster worm simulation (found in Section 6.6.4), with higher latency and much higher modeled worm growth. The format of the output is described here:

Regular Output

Every time tick in the system, the simulator outputs a line of text to indicate the current state. The output has the following fields:

"T:" indicates the current tick of the simulation

"EvCount:" indicates the number of events currently scheduled to be executed. This will increase exponentially in worm simulations, and is typically almost entirely composed of "send" events.

"TimeDif:" is the difference in milliseconds between the completion of the last tick and the one prior to its execution. This is useful in estimating the remaining time in a simulation.

Network Logging Output

Periodically, the state of the network is written to file. This serves as a backup in case of system failure during critical or lengthly simulations, as well as allowing post-execution study of the system as it executes. During this time, more information about the system is gathered and presented to the user. Due to the relatively high cost of writing large amounts of data to file, this action is only performed occasionally throughout a simulation, typically on the third, fifth, tenth, thirtieth, fiftieth, hundredth, three-hundredth (and so on) ticks.

The fields in this output are as follows:

"Worm name infections:" indicates the number of fully simulated hosts that the worm has infected.

"Worm name infections packetsink:" indicates the number of abstracted hosts that the worm has infected.

"Scheduler count:" shows the number of events currently scheduled to be executed. This will increase exponentially in worm simulations, and is typically almost entirely composed of "send" events.

"Message count:" is the number of messages currently in circulation (i.e. on incoming or outgoing queues) throughout the system.

"Message from sink count:" is the number of messages that have been sent by the abstracted network.

"Message from host count:" is the number of messages that have been sent by fully simulated hosts.

"% messages from hosts:" is the percentage of the total messages that have been sent by hosts. This is a good measure when testing and debugging various aspects of the abstracted network nodes.

Simulation Specifics

It should be noted that the "setup" event which creates all the nodes and the recurring worm event is executed at tick 3, and that the "finish" action and the "failsafe finish" action (both of which cease execution of the simulation, the failsafe used in the event that the "finish" action fails in some fashion) are added with the setup action.

The long buildup before the event count increases is due to the simulated latency of the Blaster worm.

```
T: 1 EvCount: 3 TimeDif: 0
T: 2 EvCount: 3 TimeDif: 3.774
T: 3 EvCount: 103 InfCount: 0 TimeDif: 229.122
--> Logging network state < 3 > <--
---------------------------------------------------
Current information about the system:
Blaster_Red_Infections_PacketSink : 0
Blaster_Red_Infections : 1
Scheduler count: 103
Message count: 1
Message from sink count: 0
Message from host count: 1
% messages from hosts: 100
---------------------------------------------------
T: 4 EvCount: 102 InfCount: 0 TimeDif: 12.342
T: 5 EvCount: 102 InfCount: 0 TimeDif: 0.041
T: 6 EvCount: 102 InfCount: 0 TimeDif: 0.323
```

```
T: 7 EvCount: 102 InfCount: 0 TimeDif: 0.044
T: 8 EvCount: 102 InfCount: 0 TimeDif: 0.045
T: 9 EvCount: 102 InfCount: 0 TimeDif: 0.052
T: 10 EvCount: 102 InfCount: 0 TimeDif: 0.044
--> Logging network state < 10 > <--
---------------------------------------------------
Current information about the system:
Blaster_Red_Infections_PacketSink : 0
Blaster_Red_Infections : 1
Scheduler count: 101
Message count: 1
Message from sink count: 0
Message from host count: 1
% messages from hosts: 100
---------------------------------------------------
T: 11 EvCount: 100 InfCount: 0 TimeDif: 0.476
T: 12 EvCount: 100 InfCount: 0 TimeDif: 0.04
T: 13 EvCount: 100 InfCount: 0 TimeDif: 0.044
[...]
T: 906 EvCount: 1920639 InfCount: 343205 TimeDif: 49082516.863
T: 907 EvCount: 1992617 InfCount: 355395 TimeDif: 52358467.866
T: 908 EvCount: 2067530 InfCount: 368012 TimeDif: 56816139.746
T: 909 EvCount: 2145767 InfCount: 381037 TimeDif: 59607466.733
T: 910 EvCount: 2225760 InfCount: 394569 TimeDif: 66807330.115
[...]
```

Appendix C: Simulator Software and Plugins

The software used for this research may be found at http://snrg.ict.ru.ac.za/projects/graphsim-a-robust-network-simulator/. It is developed using monodevelop[1], and executes upon the open-source project "mono"[2].

A selection of developed plugins are shown below, with development testing and debugging plugins (which are unstable) marked by italics:

Node plugins

DefaultNode A default node which does nothing

IPv4 An abstract node representing any IPv4-addressable node

IPv4Node An implementation of the IPv4 abstract node, this was used for the majority of early simulations

InfectableHost A node representing any IPv4-addressable node capable of worm infection

PacketSink An early conceptual "network node" which performed no operations on any incoming packets

IPv4_Partial_Node A 'leaf' node of a tree used for storing nodes in memory for quick access

IPv4_Full_Node A 'parent' node of a tree used for storing nodes in memory for quick access

SimpleNode A simple IPv4 node used in later simulations

Node The generic name for standard nodes used in later simulations, it could perform simple routing and was infectable by Internet worms

NetworkNode The generic name for standard network nodes used in later simulations, it accepted and logged incoming worm packets

Connection plugins

[1]http://monodevelop.com
[2]http://www.mono-project.com

DefaultConn A default connection which does nothing

ImmedConn A simple connection object used in early debugging that sent packets without any form of queue or delay in transfer

DelayConn A connection object that implemented a random delay via packet queues

IPv4Conn A connection betweeen two IPv4-addressable nodes, otherwise similar to an *ImmedConn*

Conn The generic name for standard network connection used in later simulations, it immediately sent packets, using no queueing mechanism by using the queues in the sending/receiving nodes, because latency was implemented by action modules

Packet plugins

DefaultPacket A default packet which does nothing

IPv4Packet A standard packet with IPv4 "from" and "to" address fields

Packet The generic name for network packets, these also used IPv4 "from" and "to" fields, and were very similar to *IPv4Packet*s

Action plugins

DefaultAction A default action which does nothing

BasicSend This plugin was developed in the early stages of simulation development to test the communication framework. It sends a packet using a *DelayConn*

BasicReceive This plugin was developed in the early stages of simulation development to test the communication framework. It receives a packet that had previously been sent via *BasicSend* that is enqueued on a *DelayConn*

NRSend/RNSend/NNSend/NISend These plugins were used for early routing prototyping. The prefixed letter couple uses N to represent nodes, R for router, and I for Internet (or network node). The first letter indicates the sender of the packet, the second letter the receiver of the packet.

LogEvent This plugin was used throughout the debugging phase to write detailed system states to file, enabling in-depth study of the operations of the simulator.

Setup A catch-all plugin which is overridden for each simulation, it creates the structure of the network using other plugins and queues the first round of actions that are to be executed

Finish A catch-all plugin which is overridden for each simulation, it stops the execution engine and, if necessary, write the final state of the network to an output - either file, or to the console

Worm A plugin that represents a worm. Every time this action executes, it iterates through every infected node, enqueueing worm packets that are generated according to the worm scanning algorithm specified, and then adds a new instance of itself to the scheduler in the nearby future