8-1-2005

On the Design of an Immersive Environment for Security-Related Studies

Yougu Yuan
Dartmouth College

ON THE DESIGN OF AN IMMERSIVE ENVIRONMENT FOR SECURITY-RELATED STUDIES

Yougu Yuan

Technical Report 2005-552

Department of Computer Science

Dartmouth College

Hanover, NH, 03755

ON THE DESIGN OF AN IMMERSIVE ENVIRONMENT FOR SECURITY-RELATED STUDIES

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Master of Science

in

Computer Science

by

Yougu Yuan

DARTMOUTH COLLEGE

Hanover, New Hampshire

August, 2005

Examining Committee:

(chair) David Kotz

Sean W. Smith

David M. Nicol
University of Illinois at Champaign-Urbana

Charles K. Barlowe, Ph.D.
Dean of Graduate Studies

Abstract

The Internet has become an essential part of normal operations of both public and private sectors. Many security issues are not addressed in the original Internet design, and security now has become a large concern for networking research and study. There is an imperative need to have an simulation environment that can be used to help study security-related research problems. In the thesis we present our effort to build such an environment: *Real-time Immersive Network Simulation Environment (RINSE)*. *RINSE* features flexible configuration of models using various networking protocols and real-time user interaction. We also present the *Estimate Next Infection (ENI)* model we developed for Internet scanning worms using *RINSE*, and the effort of combining multiple resolutions in worm modeling.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Security over the current global Internet is a large concern for both normal users and the research community. A wide variety of attacks or abuses are carried out constantly in the networks, ranging from spam, phishing, to large scale attacks such as distributed denial-of-service and Internet worms.

The security over the Internet is in general a challenging problem partially because that the Internet itself is a very large complex system. The Internet consists of millions of individual computing devices and terabits of traffic are delivered over the Internet every single second. The scale of the system alone is less of an obstacle when compared with its heterogeneity. The heterogeneity of the global Internet not only lies in its various computing devices and network topologies, but also in the communication traffic it carries. The most challenging property is that the Internet is an evolving system. Computer devices are added to and removed from the Internet all the time, new applications or protocols are introduced to the system and spread out quickly.

Given a complex system of that scale, the Internet cannot be studied using analytical models alone. An analytical model uses mathematics to formulate and solve problems. It is often limited because of unrealistic simplifications that may have been applied to create a tractable problem.

"A simulation is the imitation of the operation of a real-world process or system over time" [2]. In a discrete-event simulation the states of the system only changes at discrete points of time. Simulation techniques have been widely used to study the behavior of complicated systems. However, given the scale and the complexity of the Internet, one must realize that only the techniques that support models of very large scales can be used in this context.

The fact that we want to carry out security-related studies here further emphasizes the use of simulation technologies. To perform security studies, new technologies and algorithms need to be designed, the designs needs to be prototyped, the prototypes need to be tested, and all these must be carried out without disturbing the operations of real existing system. These all call for an environment that can be used to study security-related problems, understand their technical implications and assess effects of newly designed algorithms. In the thesis we present out effort to build such an environment.

We also demonstrate the use of the built environment through a case study on Internet worms. One of the major threats to the current Internet is large-scale malicious code, or computer worms. The key feature that distinguishes a computer worm from a virus is its ability to propagate with no or very little human interaction. This feature and the fact that the Internet is well-connected enable a worm to spread and infect the networked vulnerable machines in a short amount of time, and it often rules out any possible human reaction. Internet worms usually take just hours to days to propagate [6, 56], and the period can potentially be further reduced [53]. The time interval between a vulnerability being publicized and a worm exploiting such vulnerability comes out also show a trend of shrinking [48]. All above symptoms call for an immediate response from the research community to come up with effective solutions or at least mitigations. We present our research effort in modeling and simulating Internet worms. The models we develop can be used to study Internet worm behavior and exercise proposed detection and defense systems.

1.2 Contributions and Thesis Outline

The contributions of the presented research work are mainly in two parts:

- We designed and built *iSSFNet* framework, the simulation infrastructure used in *Real-time Immersive Network Simulation Environment (RINSE)*. I have also implemented many packages inside *iSSFNet*. *iSSFNet* is built on top of iSSF and supports parallel execution. It features flexible configuration of networking protocols and topologies, real-time user interaction, and packages that exercise networking attacks and defenses.

- We studied the modeling and simulation of Internet scanning worms. The *Estimate Next Infection (ENI)* model is developed inside *iSSFNet* and is validated against real-world data trace. We also combined fluid-oriented models with the packet-oriented models to achieve scalability while maintain packet level detailed information. Some further optimizations have been developed to improve the model performance.

The rest of the thesis is arranged as following: In chapter 2 we presented the effort to build *iSSFNet*. In section 2.1 a review is given on the research on network simulations. *iSSFNet* is one main component of *RINSE*, so in section 2.2 we have an overview of *RINSE*. Section 2.3 explained the design of *iSSFNet*, detailed its support for parallel execution and immersive simulation.

Chapter 3 discussed the modeling and simulation of Internet scanning worms. In section 3.1 we give some background knowledge about Internet worms. Section 3.2 introduced two traditional worm models. The *Estimate Next Infection model* is presented in section 3.3. In section 3.4 we discussed the effort of combine multiple resolutions in worm modeling. Section 3.5 we present the optimizations we used in worm simulation and other experiment results.

In Chapter 4 we give a summary of the thesis and some possible future work.

Chapter 2

iSSFNet

2.1 Simulation of Large-Scale Networks

The global Internet is a constantly changing system with millions of various computing devices. Its complexity, heterogeneity, size and rapid evolution all contribute to the fact that simulation of the Internet is an immensely challenging task [17].

Various simulators have been developed over the years to simulate large-scale networks [19, 49, 66]. One widely used network simulator in the research community is the *Network Simulator (ns-2)* [42]. *ns-2* is a discrete event simulator designed to help networking research. It has been used to study TCP, routing and multi-cast over both wired and wireless networks. Many packages contributed by different researchers greatly enriched the capabilities of *ns-2* and ensured its popularity.

ns-2 is based on a sequential discrete-event simulation engine. In order to simulate large-scale network models, the *Parallel / Distributed Network Simulator (PDNS)* [19] has been developed as an extension to *ns-2* engine. PDNS used federation approach to distribute time-synchronized events among multiple instances of the original sequential engine running on a distributed platform [46]. However, to take the advantage of the parallel extension, substantial modifications are needed to the

syntax of the network model description, which brought inconvenience to network modelers. The same kind of approach of providing *Real Time Infrastructure (RTI)* for sequential simulators, or so called *federates*, has also been used in GTNetS [43]. Using *RTI* to integrate multiple *federates* is proposed in *High Level Architecture (HLA)* standard made by the *Defense Modeling and Simulation Office (DMSO)* [37]. The original purpose of *HLA* is to promote interoperability of different simulators. When *HLA* is followed, only minimum changes are required to the original sequential simulator to make use of the standard communication and synchronization services to the participating federates. One must note, however, that there is an overhead to ensure interoperability among different simulators.

Another approach to parallelization is to develop a parallel simulator from scratch. This approach usually means that the simulator is a homogeneous system and there is no interoperability issue. The simulator can then focus on efficiency and try to achieve better performance. The *Telecommunications Description Language (TeD)* developed on top of *Georgia Tech Time Warp (GTW)* [14] is one such system. In *TeD / GTW* system, the *TeD* description is converted into executable for *GTW* and can be executed in parallel [45]. The followup research effort continued as *TeD / Nops* at Dartmouth College [44]. The *Northern Parallel Simulator (Nops)* is a process-oriented conservatively synchronized parallel simulation system. The success of *Nops* encouraged further research in the *Scalable Self-organizing Simulations (S3)* project and resulted in the invention of *Scalable Simulation Framework (SSF)*.

SSF is a compact set of programming interface [51]. It is developed to provide core support for parallel simulation of large-scale telecommunication systems. There have been several implementations of *SSF* in both Java and C++. Java version *SSF* is developed at Renesys Corporation. Using Java *SSF*, a wide range of packages have been developed for network simulation and these packages are organized into the simulator *SSFNet* [10]. The main components of *SSFNet* are (1) protocol modules such as IP, TCP, UDP, sOSPF and BGP; (2) network elements such as hosts, routers, links and subnets. Complex network models can be assembled using these basic components.

The main instance of the *SSF* C++ implementation was *Dartmouth SSF (DaSSF)*. *DaSSFNet* was developed as the C++ counterpart of Java *SSFNet* [32]. With the support of *DaSSF*, *DaSSFNet* can be used in both shared-memory multiprocessor architecture and distributed environment. *DaSSFNet* has support to fluid-oriented simulation and can simulate a mixture of packet level traffic and fluid flow level traffic. One drawback of *DaSSFNet* is the lack of routing protocol models such as OSPF and BGP. As a compensation, it can pre-load IP forwarding table information dumped using Java *SSFNet* on the same network configuration files.

The research effort on network simulator is not only focused on its scalability, but also on extending its usage. Emulation is the effort that tries to address the lack of real-world interaction in traditional network simulators. Emulation in this context can be classified into *network emulation* and *direct execution* [16].

- *Network emulation* allows the simulated components to communicate with the real-world protocol implementations. It can be further classified depending on whether both two end-points of the communication are real-world protocol implementations [4, 16, 49]. If two end-points are both real hosts, the simulated network only influence the communication indirectly through simulated topology, network delay, and activities such as congestion and dropping packets. If one of the endpoint is not real host, i.e., one simulated device communicates with a real-world protocol implementation, the implementation is supposedly more sophisticated because it involves most of the details one would encounter in a real implementation. Although the technique theoretically can be applied to any protocol, only simple ones such as *ping* and *traceroute* are supported in most such emulators.

- *Direct execution* provides an environment that the real-world code can be executed inside with minimum or none modification [34]. Different from *network emulation*, this approach does not always have real-time constraint.

The interaction between the network simulator and the real-world should not be limited to pro-

tocol implementations. In many situations, the direct interaction between the network simulator and the user is more useful. For example, while simulation is running, the user can shutdown / reboot simulated hosts, install / upgrade software on them, change traffic filtering rules, or even launch simulated attacks and deploy defenses in the simulated network. All these interactions may not require protocol models inside the simulation to be as detailed as in the real world, but they need to account for most common operations that may involve themselves in the real world. I will discuss the support for this kind of interactions more in section 2.3.3. The traditional network simulator alone clearly cannot provide such functionality. The design of an immersive conservative network simulation system needs to have the following pieces.

Support real time simulation One prerequisite of user interaction is that the simulator needs to be paced with real time. A simulation that runs too quickly does not leave the user enough time to react. On the contrary, a simulation that runs in real time or even reasonably slower than real time would facilitate user interactions and have a better effect of mimicking a real system. An desirable feature would be a simulation pacer that can be used to speed up or slow down the simulator to meet different simulation needs.

GUI front-end Although it is not necessary for an ordinary network simulation, a Graphic User Interface (GUI) front end is essential for immersive simulation. The use of the GUI is in two folds. First, the users need to monitor the simulation progress and inspect simulation output in real time. In the network simulation context, such output may include topology information, traffic over the links, status of the simulated hosts / routers, etc.. Second, the users need a tool to inject user input into the simulator as the simulation is running. In the context here such input may include actions such as bootup / shutdown devices, launch / kill simulated applications running on simulated hosts, initiate simulated attacks, etc..

Support external simulation events When the user input or reaction is delivered to the simulator, it needs to be processed and converted into a simulation event and injected into the event queue with an appropriate time-stamp.

Support run-time data collecting and export The output of the simulation and the results of the user input need to be organized and sent back to appropriate users. It is also possible that different users are interested in different data of the same simulation, so a design of an efficient distribution of the simulation output data is highly recommended.

In the rest of the chapter, I first give an overview of the research effort of our group to develop such an immersive network simulation environment, then I focus on the design and implementation of the network simulator *iSSFNet*.

2.2 Real-time Immersive Network Simulation Environment

Real-time Immersive Network Simulation Environment (RINSE), has been developed to support security related large-scale network simulations and exercises [26]. *RINSE* is part of the project *MOSES (Modeling Of Security and Systems)* [39], and it has been developed to meet the increasing need of automated tools to help study, analyze, and assess security related network activities. Traditionally emulation and simulation are two main methods used for such purposes. Emulation provides a more realistic environment but does suffer in scalability. Simulation does not have as many details but has an advantage in terms of scalability, and it by nature excludes any risk of accidental letting loose of the exercised attacks.

2.2.1 Simulation Core Support

One design goal of *RINSE* is to enable multiple users to interact with the ongoing simulation simultaneously. To achieve this goal, more simulation core support is needed. Major extensions have been added to the original *Dartmouth SSF (DaSSF)* and the new simulation kernel is named *iSSF*. *iSSF* is a C++ implementation of the *Scalable Simulation Framework (SSF) API*. Five major classes are defined in *SSF* and briefly introduced below. More detailed description of the five base classes and examples of using them can be found in [33].

Entity is a simulation state variable container. For example, in a network simulator, one might represent a host or a router using an entity. An entity can own instances of *process*, *inChannel* and *outChannel*.

Process is used to specify state evolution of an *entity*. An *entity* can have multiple *processes* attached to it. A *process* can pause itself and either wait for a specific amount of simulation time or wait on a set of *inChannels* for incoming *event*

inChannel In *SSF* a channel connects entities. A message can be delivered from the source entity to the destination set of entities through a channel. An *inChannel* is the destination end of a channel on an entity. Multiple *outChannels* can be mapped into the same *inChannel*.

outChannel is the source end of a channel. An *outChannel* can be mapped into multiple *inChannels*.

Event is a message that is exchanged between entities.

In *iSSF* two major extensions have been added in the simulation engine to support immersive simulation:

- *Simulation pacing support*. A ratio between the simulation time and real world clock time can be specified. This can be used to force real time execution as well as setting an appropriate pace of the simulation running for different needs. When the simulation cannot keep up with the specified pace, statistics of events that missed the appointment time will be collected as an indication.

- *Import / Export external event support*. A special *inChannel* is added to *iSSF*. External events such as real packets received through real sockets can be passed to the special inChannel and converted into simulation events using user customized callback function. Similarly, a special *outChannel* is added and simulation events that go through this *outChannel* will be delivered to a user customized callback function and leave the simulated world.

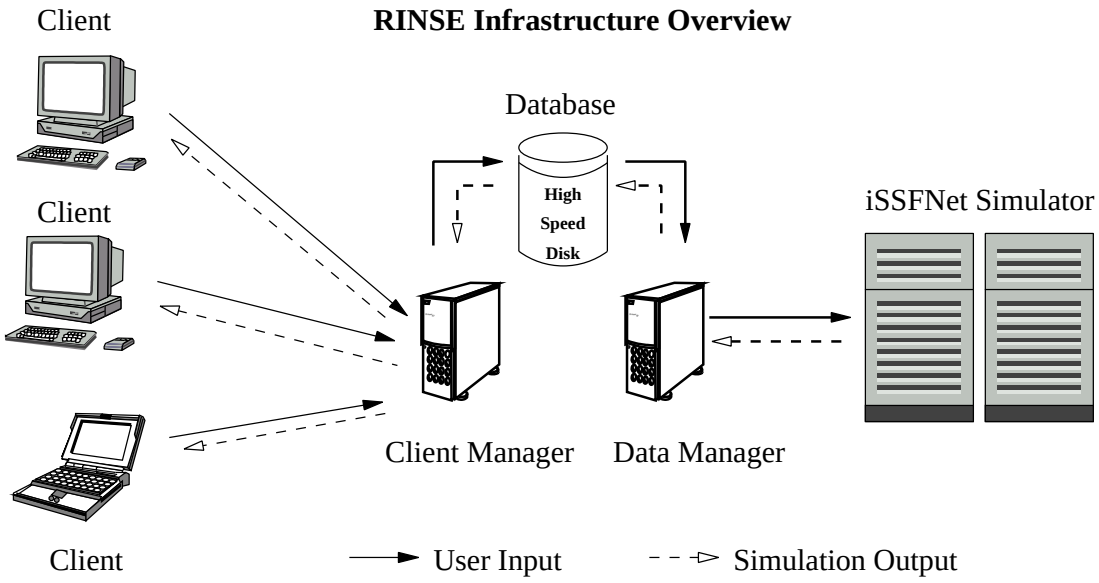

Figure 2.1: The Infrastructure Overview of *Real-time Immersive Network Simulation Environment*

2.2.2 *RINSE* Infrastructure

RINSE currently has five main components, as shown in Figure 2.1.

iSSFNet is the network simulator. It supports immersive simulation and parallel execution. More details of iSSFNet will be described in the rest of this chapter.

Data Server A server directly connect to the network simulator. It collects the outputs from the simulation and distribute them into different tables of the database. It also extracts user inputs from the database and delivers them to the simulator.

Database A database is used to store all outputs and user commands. In the future we hope the that the database can store enough trace information to support simulation checkpoint, and facilitate more efficient experiment reproducing.

Client Management Server A server manages multiple clients. Each client needs to present a user name and password. Different views are associated with different user names. In the

real world, a network admin often would just have the privilege to monitor and collect data within the subnet that he administrates; an ISP would usually only know the detailed topology of its own network and have control of the computing devices inside. This can be reflected through different views of the simulated network. In the same time, different users may want to collect different kinds of data from the simulation. An ISP may care more about the total throughput of its own part of the network, while a system admin may pay more attention to intrusion detection and system health. The users specify the data they are interested in through subscription to this server, and the server needs to collect information from the clients, extract data from the database and deliver the data to the corresponding clients.

Client This is a GUI front-end. It provides an interface for different users to view different parts of the simulated network, monitor different data that they are interested in, and allow the users to input commands to alter the simulated network activities. In the future, web-based GUI may be developed for *RINSE*.

2.3 *iSSFNet* Design

iSSFNet is built on top of *iSSF*, which supports both the shared-memory multiprocessors and distributed environments. *iSSFNet* modules are also parallelizable, and *iSSFNet* is designed to simulate security related large-scale network activities.

Many network elements in the real world can find their counterparts in *iSSFNet* modules. In the real world the network protocols are used to define the transactions between networked devices, and the organization of those protocols follow a fairly clean layered pattern. In *iSSFNet* the *ProtocolSession* class is used to model a protocol, and multiple sessions are organized into a graph, *ProtocolGraph*, to simulate the protocol stack in a host / router in the real life. The interactions among different sessions are defined using limited number of well-constructed APIs. A *ProtocolGraph* and one or more network interface cards (NICs) are the main components of a host or a

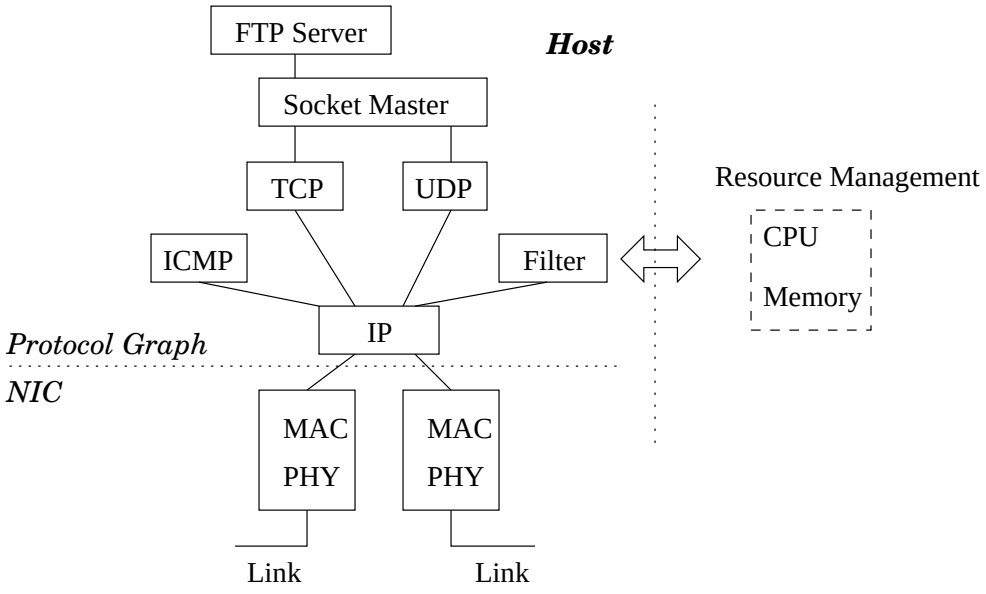

Figure 2.2: Internal structure of an example host

router. A NIC is also a *ProtocolSession* stack. For now we only have simplified Media Access Control (MAC) layer and physical (PHY) layer, but the design would easily enable future elaboration of the models of these two layers, or even enable wireless extension. A host / router can also have a model for CPU and memory. APIs are also defined to allow any protocols to influence the CPU and the memory utilization as well as query the utilization and change behavior accordingly. For example, a router may decide to drop packets because the CPU is too busy to keep up forwarding the incoming packets. An example internal structure of a host is show in Figure 2.2. It has two network interface cards as well as a protocol graph that includes IP, TCP, UDP, Socket, and an application layer FTP server.

The NICs of hosts / routers can be connected using links, thus subnets are formed. Connecting subnets can form higher-tier subnets. Repeating this process can create simulated network of very large scale. All the configurations of the simulated network are written in *Domain Modeling Language (DML)*, and I will describe *DML* in more detail in section 2.3.2.

Many networking protocols have been implemented in *iSSFNet*. Studying Internet worm behav-

ior using *iSSFNet* can benefit from the abundance of network protocols and other supports. Table 2.3 listed most of the important high level modules in *iSSFNet*.

2.3.1 Support for Parallel Execution

iSSF supports both shared-memory multiprocessors and distributed platforms. The details of the underneath architecture is hidden from *iSSFNet*. The design of *iSSFNet* also tries to hide any details of parallelization from developers of individual protocols.

Hosts and routers in the simulated network are divided into *provinces*. A *province* in *iSSFNet* is a *timeline*, or a logical process (LP) in parallel discrete-event simulation terms. All the simulated devices in the same *province* have consistent simulated clock throughout the simulation run. At any wall-clock time, the simulated time in two *provinces* may be different. Delivering packets among simulated hosts / routers within the same *province* are implemented using timers and function calls. While packets go across to a different *province* are delivered as simulation events through SSF channels. There may be multiple simulated links between two *provinces*, but they share the same SSF channel using the minimum link delay as the channel delay. The extra delays are added after the packets are delivered to the other *province*. All these designs are to reduce the number of simulation events and minimize the synchronization overhead.

One of multiple *provinces* are grouped into a *country*. Each *country* is a process and all *provinces* in the same *country* share the same memory space. In another word, different *provinces* in the same *country* can share time-invariant state variables in the simulation. In a shared-memory multiprocessor architecture, the *countries* are then allocated to different CPUs; while in distributed environments, they are allocated to different platforms. Figure 2.3 illustrated how a simulated network may be divided when the simulation is run in a distributed environment.

Table 2.1: *iSSFNet* Modules

Module	Explanation
IPv4	Internet Protocol implementation
ICMP	Internet Control Message Protocol
TCP	Packet oriented TCP implementation
Fluid TCP	Fluid oriented TCP implementation
UDP	Packet oriented UDP implementation
Fluid UDP	Fluid oriented UDP implementation
sOSPF	Static OSPF implementation
PAO	On-demand policy based routing support
Apps	Various application level traffic models
Background Traffic	Model for background traffic (using fixed point computing)
DDoS	Model for Distributed Denial of Service attacks
Worm	Model for Internet worms
Simple MAC & PHY	Implementations of simplified MAC and physical layer
Filter	Simplified filter implementation, can be used to simulate firewall
Fluid	Provides support for fluid oriented simulation (including fixed point)
Report	Provides support to systematic status report and data output (to the database)
Interact	Provides support to real time user interaction
Interface	Model for Network interface card
Host & Router	Model for hosts and routers
Link	Model for Network link
Net	Model for a subnet

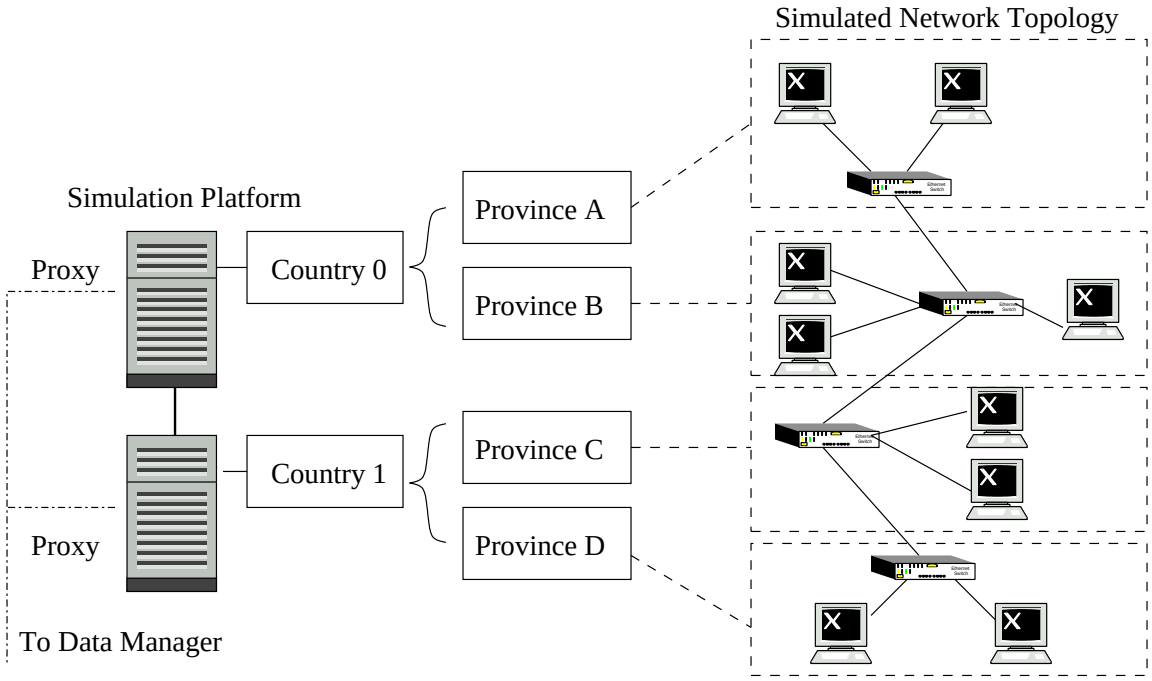

Figure 2.3: Support parallel execution in *iSSFNet*

2.3.2 Domain Modeling Language

The *Domain Modeling Language (DML)* is a "public-domain standards for attribute databases for model configuration and verification" [51]. *DML* has also been used in Java *SSFNet* and *DaSSFNet* to specify network topology. It is also used in *iSSFNet* to maintain backward-compatibility. When *iSSFNet* is started, one or more *DML* files are specified in the command line, and they are used as input to build the network model and initialize it.

The essence of using *DML* in *iSSFNet* is to have a simple database that stores the network configuration. The syntax of *DML* is very simple. A *configuration* written in *DML* is enclosed by brackets and composed of *(key, value)* pairs. The *value* itself can recursively be a *configuration*. A simple segment of *DML* is given in List 2.1. This *DML* segment describes a network that has two hosts inside, and the two hosts are connected to each other through a point-to-point link.

Listing 2.1: Simple *DML* example

```
Net [
  host [
    idrange [ from 0 to 1 ]              # two hosts

    # substitution
    _find .dictionary.workstation.graph

    interface [ id 0 ]
  ]
  link [ attach 0(0) attach 1(0) delay 0.1 ]
]
```

The '#' in *DML* is the symbol for comments. The two hosts have an id "0" and "1" respectively, and each has a network interface card with id "0". The link specifies its endpoints using "attach" keyword. "0(0)" inside means "host 0, interface 0"; similarly "1(0)" means "host 1, interface 0". This addressing scheme is called *Network-Host-Interface (NHI)* addressing. If one assign an id to the specified "Net" here in this *DML*, then it can be used as a subnet in a larger topology. If an id "1" has been assigned to this subnet, then the complete *NHI* address of host 1, interface 0 would be

"1:1(0)". Adding another level of subnet would just add and id and a ':' before the original *NHI*.

DML has two often used special keywords: `_find` and `_extends`. `_find` is used to do inline substitution. The line `_find .dictionary.wormstation.graph` in the example *DML* segment means that the *graph* of the host is detailed inside ".dictionary.wormstation.graph". This is another addressing convention in *DML*, and it refers to a segment of *DML* that can be found in in List 2.2. The protocol graph inside a host, in this example, has IP, ICMP, TCP and Socket layer installed.

Listing 2.2: Simple *DML* example (cont.)

```
dictionary [
  workstation [
    graph [
      ProtocolSession [ name socket
                        use SSF.OS.Socket.socketMaster ]
      ProtocolSession [ name tcp
                        use SSF.OS.TCP.tcpSessionMaster ]
      ProtocolSession [ name icmp
                        use SSF.OS.ICMP.ICMPSession ]
      ProtocolSession [ name ip
                        use SSF.OS.IP ]
    ]
  ]
]
```

`_find` can greatly reduce the size of the *DML* files because it enables re-use of the *DML* block, but it does not allow addition of attributes to the substituted block. `_extends` can be used in such situations to provide inline inheritance. The line `_find .dictionary.workstation.graph` can be replaced by the *DML* in List 2.3. Then additional attributes can be added to the graph if needed.

Listing 2.3: Simple *DML* example (cont. 2)

```
graph [
  _extends .dictionary.workstation.graph
]
```

With inline substitution and inheritance, *DML* can be fairly efficient when used to build network models of very large scale. All the keywords except for the special ones are defined by the user, which makes *DML* very flexible and expressive.

2.3.3 Support for Immersive Simulation

iSSFNet has support for *immersive simulation*. The term *immersive simulation* is often used in the context of *Virtual Reality (VR)* which is out of the scope of this thesis, but the fundamental idea of *immersive simulation* here in *iSSFNet* is analogous to its meaning in *VR* context — to develop a simulation system that the user can interact with as if the user were interacting with a network in the real world.

When only the supports (event scheduling, real-time synchronization, etc.) from the simulation engine are considered, the techniques needed are almost the same in *immersive simulation* as in emulation. In fact, we did use the same kernel support to implemented the emulation support in *iSSFNet* [26]. However, *immersive simulation* is different from the traditional emulation in the high level.

- Traditional emulation either interacts with real-world protocol implementation or provides an environment for direct execution (section 2.1). Immersive simulation in *iSSFNet* emphasizes on human-in-the-loop interaction. Real-world code implementation is not required to provide such interaction support. Some common operations on a host / router are not specifically related to any particular single protocol. For example, real-world events such as router shutdown and network link failure take place constantly in the real Internet. Some other operations do not have real-world equivalent but are also useful to have in the simulation: one may want to adjust the amount of data trace collected from the simulation, or turn on / off monitoring on a specific simulated host / router. These kinds of interactions cannot be easily exercised either through interacting with real-world protocol implementation or direct execution. There are also security-related operations that are too risky to exercise using real-world

code implementation or do not have a very good counterpart in the real world, e.g., ddos attack, worm attack, buffer overflow, etc..

- The infrastructure most suitable for this kind of human-in-the-loop interactions is different from the ones in traditional emulation. The *direct execution* type of emulator needs a wrapper that can intercept system calls from the real-world protocol implementation and provide a replacement for compiling, linking or run-time calling [34]. The emulator that can interact with real-world protocol implementation usually has some special agents inside the simulated network. A special agent connects to a real host through a special channel. This channel can be treated in two ways: it may be considered as a link in the simulated network, thus the real host are *connected* into the simulated network through the special link; it may also just be considered as an internal data structure so that the special agent just represents the real host in the simulated network. For human-in-the-loop interactions, the above infrastructures are not suitable. Technically these interactions we desire must be allowed to take place between the user and any of the simulated host / router, or even between users and some protocol sessions inside a simulated host / router. This means that it needs an infrastructure that can receive the operation commands, distribute them to the appropriate part of simulated network, and send back possible feedback. Fortunately, the occurrence of these interactions normally would not be as frequent as packets in a real-world traffic flow, e.g., FTP file transfer, so not too much overhead would be introduced for the command distribution.

In this section I introduce the commands currently implemented in *iSSFNet* and the infrastructure that supports command processing and distribution.

Command Message

Users can interact with *iSSFNet* using *commands*. The *commands* in *iSSFNet* have some general fields:

```
<cmd name> <exec host> {<parameters>}
```

The *command name* can uniquely identify one command. The second field is the name / NHI / IP address of the host or router that the command needs to be executed on. Below is an alphabetically ordered list of the commands implemented in *iSSFNet*.

bootup is used to turn on the specified host / router in the simulated network.

```
bootup <host>
```

create_exploit is used to declare a type of *exploit* from a given *DML* configuration file. The created exploit can be used later to compromise simulated nodes. An exploit usually specifies the vulnerable OS, application, version number, etc., and a simulated node may be compromised or crashed by the exploit if its OS and application are vulnerable to it. The exploit may also have an embed command. If a vulnerable host is compromised, this embed command will be tried on the compromised host. In general any user command can be used as this embed command.

```
create_exploit <DML file>
```

ddos_attack is issued to a ddos session inside a simulated host. It launches a Distributed Denial of Service (DDoS) attack from the executing host against the target host. The parameters specify the duration of the attack and the average attacking traffic rate from each participants (*zombies*) of the attack. The DDoS attack model implemented in *iSSFNet* carries out the attack mainly by squeezing target bandwidth or CPU / memory utilization.

```
ddos_attack <src> <dst> <duration> <avg_rate_kbps>
```

filter is issued to the FilterSession on a simulated node. It can turn on/off the filters, or list the filters currently in the vector. It also allows users to add / remove a filter to / from a specific

position of the vector. Each filter can be specified by a *filter_str*. A *filter_str* specify a tuple of `(interface, protocol, src_ip, src_port, dst_ip, dst_port)` and an action (`allow` or `deny`) for the packets that satisfy the tuple.

```
filter <host> list
filter <host> on / off
filter <host> remove <index>
filter <host> add <index> <filter_str>
```

ftp_get is issued to a tcp client session. Upon receiving it, the client should initialize a FTP file transfer from the specified simulated server, using the given port number.

```
ftp_get <client> <server> <port> <file_size>
```

ping is issued to ICMP session. The node that executes this command sends `repeat_num` of ICMP echo request messages to the destination host. The response to the command is similar to the real ping command.

```
ping <src host> <dst host> <repeat_num>
```

report is issued to turn on / off the data trace collecting on the given simulated node.

```
report <host> on/off
```

setup_prot is issued to a host / router. It dynamically instantiates a new protocol session on that simulated node, and the detailed configuration is in the *DML* file(s) specified in the parameters. When carried inside an *exploit* and sent out to a vulnerable host, this command can be used to effectively simulate how the hosts are compromised in the real world.

```
setup_prot <host> <DML file>
```

shutdown is issued to a host / router to power it off.

```
shutdown <host>
```

snd_exploit is issued to an exploit session. Upon receiving it, the simulated node will send a packet that has an attached exploit. If the simulated target node is vulnerable, the exploit will be executed on the target node.

```
snd_exploit <src> <exploit_id> <target> <target port>
```

throttle is issued to change the frequency of the periodical reports from the simulation to the *data manager server* of *RINSE*.

```
throttle <new time interval>
```

worm_start is issued to the Internet scanning worm session on a host, upon receiving this, the worm session will start scanning using pre-configured scanning pattern.

```
worm_start <host>
```

Command Proxy and Command Processing

Each *country* in *iSSFNet* has a *command proxy*. When a user command arrives at the *data manager server* of *RINSE*, it will be broadcast to all *countries* using UDP packets. A possible future optimization is to send the command only to the country that needs to receive it, but this requires that the *data manager server* knows how the simulated network model is divided and mapped to different countries.

When a command arrives at a *country*, it is received by the *command proxy*. The proxy has a special *inChannel* and a special *outChannel* (section 2.2.1), each of which is associated with a real-world socket. The proxy itself is running as a separate *SSF timeline* and has channels connecting with all other *provinces* in the same *country*. The processing of the incoming command is conceptually straightforward and can be illustrated using the pseudo code in List 2.4.

Listing 2.4: Command Proxy Pseudo Processing Code

```
while (true) {
  wait for incoming command

  convert command into a simulation event: cmd
  if (conversion failed) {
    attach error message;
    send it to the special outChannel as feedback;
    return;
  }

  if (cmd is for the whole simulator)
    broadcast cmd to all provinces;
  else if (exec_host of cmd is in this country) {
    find appropriate province p;
    send cmd to p;
  }
}
```

Once the command is delivered to a *province*, the *province* identify the host / router that the command should be issued to, and distribute it through function calls. Further distribution may be needed within a simulated host or router because the command may be issued to the host / router, or an interface of this simulated node, or a particular protocol session of the node. All the distributions are implemented using well defined APIs. The design goal of the infrastructure is to (1) facilitate easy implementation of any new user commands; (2) have a centralized command distribution system.

Analytical Mode and Scripted Action

iSSFNet has support for immersive simulation, but it also keeps the flexibility of running in analytical mode, i.e. without real-time constraint. A boolean switch `support_interaction` in the *DML* configuration file controls which mode the simulation would use.

Listing 2.5: Example Interaction Script

```
scripted_actions [
  action [ time 1.5  cmd "ping client server 3" ]
  action [ time 2.0  cmd "ftp_get client server 21 50000" ]
]
```

To maintain reproducibility, the user would has the option of having the interactions scripted using *DML*. List 2.5 has an example of a script. It schedules a host named *client* to send three echo requests to *server* at time 1.5 second, and to initialize a file transfer request from the *client* to the *server* at time 2.

Chapter 3

Modeling and Simulation of Internet Scanning Worms

A computer worm is "a computer program that self-propagates across a network exploiting security or policy flaws in widely used services"[61]. With a more practical and general defnition, the computer worm can be classified as *e-mail worms*, *windows file sharing worms*, and *traditional scanning worms*[25]. In this thesis I focus on the propagation dynamics of the *traditional scanning worms*. In the rest of the thesis, the term "computer worm" refers to *traditional scanning worms* unless specified otherwise.

Hypothetically if there is a computer that has enough computing power, the most straightforward way of simulating Internet worm propagation is to build a simulated network that has as many simulated devices as the real Internet does, simulating every single packet, i.e. a packet-level detailed simulation. However, this approach inevitably has to overcome the obstacle of scalability. Simulating a network of that scale itself is a research topic [9, 11, 12], especially under the constraint of moderate amount of computing resources. Two approaches can be taken to alleviate the scalability problem.

- Reduce the scale of the simulated network. The scale is in terms of the topology, the number of vulnerable hosts, and the total IP space. For a packet-level simulation, reducing any above factors will effectively reduce the number of events in the simulation. This approach is also called "scaledown". Weaver, et al. studied the impact of applying *scaledown* techniques [59]. They tried to approximate global Internet worm dynamics by shrinking the "effective size of the network", in their study, they encountered two artifacts even after applying compensations to the *scaledown* techniques: there exist noticable biases that speed up the worm propagation, and there is also an increase in the stochastic effects. Both artifacts are significant in terms of the worm propagation dynamics.

- Use a more abstracted model in both the network topology and traffic representation. In the abstracted network topology, some nodes each represents a subnet instead of a single host / router. For the ease of description, I use the term *wormnet* for such nodes. A *wormnet* is a subnet that has hosts vulnerable to the simulated attack, and is represented by a router in the abstracted topology.

In this chapter I present the research focused on the modeling and simulation of the Internet scanning worms. Section 3.1 gives background knowledge on Internet worms. In section 3.2 we introduce two traditional worm propagation models. Section 3.3 presents the *Estimate Next Infection (ENI)* model we used in *iSSFNet*. Section 3.4 presents the work that combines fluid-oriented worm modeling with packet-oriented worm modeling. Section 3.5 discusses related problems that help complete the worm simulation and presents some other experiment results.

3.1 Large-Scale Malicious Code

3.1.1 Internet Worm History

The history of computer worms goes back to more than a decade. The first well-known large scale Internet worm incident was the Morris worm in 1988 [15]. The Morris worm has many properties that still exist in many of the newest worms. It exploited multiple vulnerabilities, propagated among multiple platforms, used local topology information in the propagation, and tried to evade detection and hinder code analysis.

After Morris worm incident, it was almost a surprise that no major worm incidents took place for another 10 years. The next Internet worm incident was in 1998 — the ADM worm appeared and introduced random scanning which has been extensively used in the later worms. In 2001, there was an outbreak of the Internet worms. There were ten recorded Internet worm incidents took place that year, including Code Red v2 [56] and Nimda [6].

Code Red v2 was the first significant Internet worm that propagates on Windows platforms. It utilized random scanning, attacked a vulnerability inside the *Windows IIS* systems and infected around 360,000 hosts in the total. Nimda worm is the most versatile worm that is known of. It is the only worm that carries four attacking vectors: e-mail, windows file sharing, web page, and scanning. Code Red and Nimda made headlines, and had significant aftermath besides reported millions of dollars damage. It has been reported after several months that there are still scans generated by infected hosts [65].

Slammer/Sapphire is the next worm incident that caught the attention of the researchers. On January 25th, 2003, with an exploit on Microsoft SQL server vulnerability, Slammer infected the majority of the vulnerable hosts within 10 minutes. It is by far the fastest computer worm [38], and its propagation is only limited by the bandwidth available to the infected machines. Slammer made it very clear that a defense system against the future worms cannot depend on human intervention.

Large Internet worm incidents after *Slammer* include *Blaster*, *Witty*, *Sasser*, etc..

Dissect Internet Worms

Having some understanding of the Internet worms behavior is a prerequisite for modeling and simulating large-scale malicious code. An Internet worm usually needs three steps to complete its propagation cycle. (1) Look for / select a target that is potentially vulnerable. (2) Try to gain complete / partial control over the selected target if possible, implant the worm program. (3) Activate the worm program. The first step distinguishes the Internet worms from the traditional viruses. It is also the step that needs to be carefully reproduced in simulation for the purpose of exercising worm detection and defense techniques.

Routing data shows that the IPv4 space is not very densely populated as of now. It is estimated that around 30% of the IP space has been advertised at the BGP[1] level. The schemes a worm can use are *scanning*, *external target list*, *internal target list* and *contagious* [60]. The majority of the existing worms belong to scanning worms. The other schemes are either only hypothetical or only have very few instances [54]. In this thesis I will focus on the modeling and simulation of scanning worms.

Scanning is by far the most common means used by the Internet worms. By carefully coding the random number generator and selecting random seeds, a worm randomly chooses an address in the 2^{32} IP space as a potential target. It is relatively simple and can achieve a good propagation speed. Several "optimizations" can be applied to scanning. This first obvious optimization is to increase scanning speed, and the extreme of that will result in bandwidth-limited worms such as *Slammer*. In preferential scanning (Code Red II, Nimda), the worms have a preference to scanning local networks, which renders faster propagation within that part of the network. Permutation scanning has been discussed by Staniford in [53] to exploit more coordinations of the worms. Wu [64] and Zou [69] discussed the hypothetical worms that utilize publicly available BGP routing table information to shrink scanning space. As shown by the past worm incidents, scanning worms have a slow start phase. To shorten this phase, one possible optimization is to pre-generate a target list

[1]Border Gateway Protocol (BGP) is the inter-AS routing protocol used throughout the Internet.

(hit list) or start the worm from multiple sources.

Although different scanning worms can utilize various exploits, target different platforms, and vary their scanning strategies, an infected host will have the following similar behavior:

- it generates many new connection attempts;
- many of its connection attempts will fail, given the percentage of advertised IP space;
- it replicates of the exploit code and sends it to infect new hosts.

3.1.2 Effort in Internet Worm Detection and Defense

In this section some possible approaches that people use to develop Internet worm detection and defense systems are discussed. These approaches have some implications on how the Internet worm should be modeled and simulated.

As stated in section 3.1.1, all scanning worms have similarities in their behavior. Thus many currently proposed Internet worm detection techniques try to take advantage of this by detecting such behaviors. A success in this kind of approach can effectively detect a class of scanning worms. Usually the techniques proposed approach the problem with one or multiple of the following means:

Egress Connection Detection This kind of approach mainly aims to detect and help defense against scanning worms within an enterprise or and institute. Williamson [63] proposed to limit the number of new connection requests that can be initiated from each end host in a given time slice. In their implementation [57] any outstanding requests will be queued and delayed. When the queue length exceeds the upper limit, no new connection requests can be sent from that host. Staniford [52] used the same heuristic but instantiated the defense in the network, which prevented more sophisticated worms from disabling any “anti-worm” software on the local host.

Jung et al. [20] proposed a port scan detection algorithm based on *Threshold Random Walk*

(TRW), which detects scans based on the success ratio of new connection requests. Weaver [62] used it in the effort to contain scanning worms in an enterprise network, and could even detect worms with very slow scanning rate (1/minute).

There are some variants of this type of detections. In *Mirage Networks* [36] and *Forescout* [18] [2], they closely monitor unused IP space to detect scanning worms and contain internal infectives.

Ingress Scan Detection The general idea of this approach is to monitor one or multiple chunks of IP space, and infer worm attacks from the observed data in the monitored space. Many proposed detection schemes fall into this category. Zou et al. [67] used *Kalman filter* (Kalman, 1960) to detect the exponential increase in the number of scans to detect scanning worms. Liljenstam et al. [28] examined the correlation of ICMP host unreachable packets to detect scanning worms. Telescope project [55] monitors a segment of unused IP space and can collect worm-scanning data. Wu et al. [64] also proposed to monitor unused space to detect scanning worms.

The major drawback of this kind of approach is that an attacker can either evade the detection by ignoring the monitored space or raise false alarms by spoofing scanning packets just to the monitored space.

Other Traffic Pattern Detection For a given worm, the packets that contain the exploit code will be sent to all possible targets. Singh [50] proposed to compute *Sampled Rabin Fingerprints* over portions of a packet, use the fingerprints and the destination port to identify suspicious traffic flows and examine the number of *(source, destination)* pairs to identify scanning worms. This algorithm may have potential usage in detecting other kinds of worms as well. Chen and Heidemann [7] proposed a different scheme to detect very fast scanning worms (scan rate $>$ 25/sec), by checking the correlation of ingress and egress destination ports, then using outgoing destination address counts as auxiliary information.

[2] Mirage Networks and WormScout are commercial products that protect enterprise networks against scanning worms.

Table 3.1: Internet Worm Detection Approaches

Pkt Level Info	Approaches
Yes	Port Threshold Random Walk [20, 62]
	Monitor correlated ICMP backscattering [3, 28]
	Detect duplicated contents [50]
No	Count initiated network connection attempts [63, 57, 52]
	Monitoring unused IP space [36, 18, 55, 64]
	Detect exponential increase of scans [67]
	Detect ingress / egress traffic port correlation [7]

Some of the above detection approaches require detailed packet level information, while for some other ones the communication pattern information will suffice. Table 3.1.2 classified the detection approaches based on whether packet level detail information is necessary to experiment such approaches.

The research work in programming languages, operating systems, and compilers on vulnerability reduction, software fault isolation, and general exploit prevention [60] is of importance but not in the scope of this thesis. The currently proposed defense strategies against Internet worms are different in their aggressiveness. Benign approaches often involve rate-limiting suspicious traffic or its connecting attempts[57, 63, 68]. The more traditional approaches include filtering or IP blocking. The most aggressive approaches might be utilizing active “counter-worm” [27].

The testing of all the mentioned defense strategies does not necessarily need detailed packet level information.

3.2 Traditional Epidemiology Models

The computer worms are analogous in many ways to the contagious disease. They both self-reproduce and actively propagate among vulnerable populations. The mathematical study of the spread of the contagious diseases can be traced back to more than three centuraries ago, and some of the epidemic models can be used to effectively study the Internet worm propagation. The contagious disease spread when some kind of *contact* takes place between the infected individual and the vulnerable individual. This *contact* cannot be easily quantified and recorded when the traditional models are used to study the spread of disease. In the instance of computer worm propagation, this kind of *contact* can be easily isolated, quantified and recorded, and this kind of contact happen through worm scanning packets. This makes it possible to re-write some formulas of the traditional models to facilitate computation. In this section two traditional models are explained:

- The *General Epidemic* model is also called *Susceptible-Infection-Removal (SIR)* model. It has been widely used to study the spread of biological viruses in *homogeneous systems*.
- *Chain Binomial* model is another commonly used traditional model, and I have implemented this model in *iSSFNet* as well.

The pros and cons of these two models will be briefly discussed, then in the next section I will introduce the model developed in *iSSFNet*, the *Estimate Next Infection (ENI)* model.

In epidemiology, it is common to just study the spread of the virus propagation over the whole system in a high level. However, this kind of approach may not suffice the requirements of some studies on Internet worms because

- it simply depends on the analytical model and ignores important factors such as network topology, other network traffic and possible operations on the computing devices inside the network.
- having a centralized module is potentially an obstacle to parallel execution.

Table 3.2: Notations in epidemiology.

Notation	Definition
$S(t)$	Number of susceptibles at time t.
$I(t)$	Number of infectives at time t.
$R(t)$	Number of immunes at time t.
C_j	CIDR space size of subnet j.
γ	Rate at which the infectives turn into immunes.
ρ	Relative removal rate.

Therefore, the models discussed in the thesis are based on *interacting groups* [13]. The intuition is straightforward: the individuals in the system is divided into groups, and each *wormnet* has a group attached to it; the worm propagation within each group is computed individually; groups can influence each other through the means of *contact* (worm scans in this context).

Before explaining the models, I first introduce some of the epidemiology notations [13] in Table 3.2. These notations will be used in the rest of the thesis, and some detailed explanations will be added later in place where they are used.

3.2.1 *General Epidemic* Model

The *General Epidemic* model is also called *Susceptible-Infected-Removal (SIR)* model. It has been widely used in the study of biological virus or contagious disease. The principles of this model was first noticed by Hamer in 1906. Kephart and White used the traditional general epidemic model to study the computer virus propagation [21, 22] They found that the *SIR* model is not accurate because the traditional virus propagation does not happen in a *homogeneous system*. The *Law of Mass Action* is the center of the traditional generic epidemic models. It states that "*for a homogeneous system,*

the rate of a chemical reaction is proportional to the active masses of the reacting substances". A *homogeneous system* means that every individual in the system has an equal opportunity to contact any other individuals in the system within a given (short) interval of time [13], which turns out to fit the propagation pattern of uniform scanning worms. In [29, 28, 53, 58, 67] the authors all used either the traditional generic *SIR* epidemic model or some extension of it to model the propagation of scanning worms. The simulation results of such models fit the collected data from real-world worm incidents pretty well

An individual in the generic epidemic system is in one of the three states: *susceptible*, *infective* or *removal*. A susceptible individual is vulnerable to the attack but not compromised yet; a compromised individual is in the state of infective, and an individual that is immune to the particular attack is called a *removal*.

In *SIR* model, the time is divided into discrete time steps. Assume that at time t, the number of *susceptibles* is $S(t)$, the number of *infectives* is $I(t)$, and the number of *removals* is $R(t)$, we then have:

$$n = S(t) + I(t) + R(t) \tag{3.1}$$

$$S(t+1) - S(t) = -\beta I(t) S(t) \tag{3.2}$$

$$I(t+1) - I(t) = \beta I(t) S(t) - \gamma I(t) \tag{3.3}$$

$$R(t+1) - R(t) = \gamma I(t) \tag{3.4}$$

where n is the total number of individuals in the system, β is the *pairwise infection parameter*, and γ is the rate of patching. If one assumes that in a given discrete time slice t, no susceptible individual will be hit be worm scanning packets twice, the β can be approximated using $\beta =$

$(scan\ rate)/C$, where C is the CIDR[3] space size [4], i.e. here $C = 2^{32}$.

Given (3.2) (3.4), we can get

$$\frac{(S(t+1) - S(t))}{(R(t+1) - R(t))} = -(\beta/\gamma)S(t) = -S(t)/\rho \tag{3.5}$$

where $\rho = \gamma/\beta$ is defined as the *relative removal rate*. According to the *Kermack-McKendrick Threshold Theorem*, A major outbreak occurs if and only if $S(0) > \rho$.

When interacting groups are considered, assume that we have groups $0, 1, ...G$, for group j, at time t, we can modify (3.2), (3.3), and get

$$S_j(t+1) - S_j(t) = \sum_{i=0}^{G} -\beta_{ij} I_i(t) S_j(t) \tag{3.6}$$

$$I_j(t+1) - I_j(t) = \sum_{i=0}^{G} \beta_{ij} I_i(t) S_j(t) - \gamma_j I_j(t) \tag{3.7}$$

where β_{ij} is the *pairwise infection parameter* between group i and group j. Note that the order of i and j matters here. β_{ij} can again be approximated using $\beta_{ij} = (s_{ij}/C_j)$, where s_{ij} is the scan rate from group i to group j, and C_j is the CIDR block size of group j.

3.2.2 *Chain Binomial* Model

The *Chain Binomial* model was firstly introduced by Reed and Frost in 1920's. Chen et al. used a model in [8] to simulate worm propagation, and that model is essentially a *Chain Binomial*. The original *Chain Binomial* model states: at time t, if the probability of a *susceptible* avoiding the contact from an *infective* is α', then the probability it avoids any contact from $I(t)$ *infectives* is $\alpha^{I(t)}$. The number of *susceptibles* at $(t+1)$ follows a binomial distribution based on the above

[3] Classless Inter-Domain Routing (CIDR) is a new addressing allocation scheme for the Internet that allow more efficient allocation than the old class A, B and C address scheme. A CIDR block can be represented using xxx.xxx.xxx.xxx/xx form, e.g., 198.162.0.0/24 represents a CIDR block of with 24 bit prefix.

[4] Note that in *SIR* model the scan rate is $scans/(time\ step)$.

probability and the number of current *susceptibles*, i.e., $S(t+1) = Binomial(S(t), \alpha^{I(t)})$.

Given the Internet worm context, this can be re-written and elaborated: For group j at time t,

$$p = (1 - 1/C_j)^{r'_j(t)} \tag{3.8}$$

$$S(t+1) = Binomial(S(t), p) \tag{3.9}$$

$$I(t+1) = I(t) + S(t) - S(t+1) - \gamma I(t) \tag{3.10}$$

$$R(t+1) = R(t) + \gamma I(t) \tag{3.11}$$

where $r'_j(t)$ is the number of scans received by the group at time t.

3.2.3 Issues with Traditional Models

The *general epidemic* model is simple and straightforward. It is a deterministic model that can easily depict high level propagation progress of uniform scanning worms. However, there are some problems with this model. The worm propagation is mainly controlled by the *pairwise infection parameter* β. When using *interacting groups*, we need to set an array of β_{ij}. If the simulated worm does uniform random scanning, the β_{ij} may just use the same value for all group pair (i, j). Different values have to be used for β_{ij} , however, when the worm does more than uniform scanning, e.g. local preferential scanning. In the original *general epidemic* model, the β_{ij} are constants. This makes it difficult to incorporate important factors such as network topology, traffic congestion, and possible defenses. The importance of including such factors can be demonstrated with the experiment below.

An artificial network topology is generated using *Boston University Representative Internet Topology gEnerator (Brite)* [35], and I attached 256 *wormnet*s to it, each *wormnet* owns a $/10$ IP prefix to form a total of /2 advertised IP space. A worm propagation scenario is exercised with 75,000 susceptibles, one initial infection, and a uniform scanning rate of 4,000 scans/sec. Two experiment runs are carried out deterministically. Traffic congestion is considered in one run but

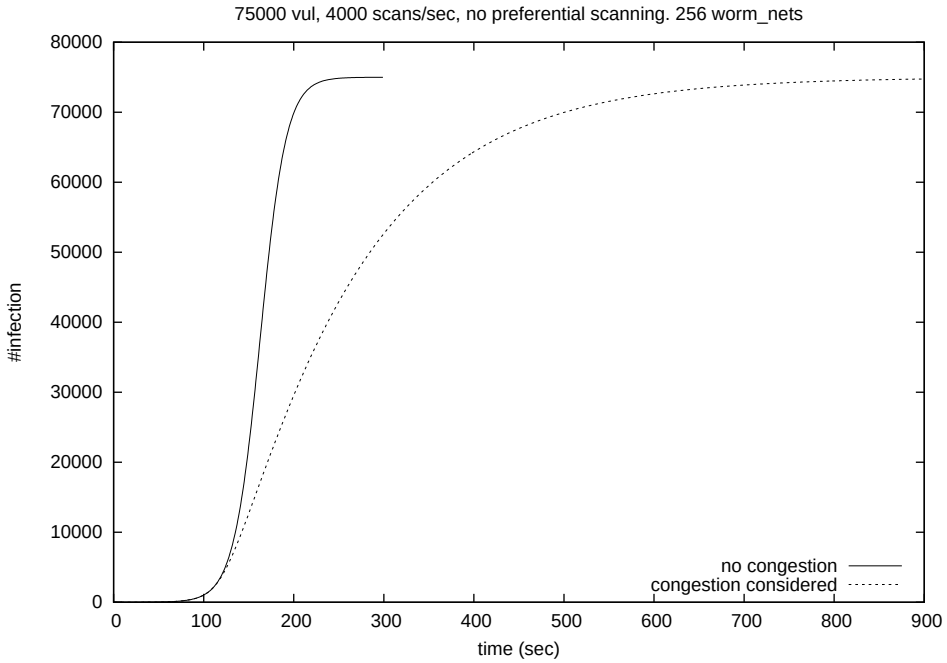

Figure 3.1: Compare Internet worm propagation with and without congestion.

not the other. As shown in Figure 3.1, the effect of traffic congestion on the worm propagation in this scenario is obvious, and ignoring such effect generates incorrect result.

The *Chain Binomial* model can be used to easily include those important factors. The worm scans sent out go through the underneath topology, experiencing possible congestion, filtering or other regulating, and arrive at the destination *wormnet*. The received number of scans $r_j^{'}t$ at the receiving end is the result after including all those network based factors, and automatically the influence is brought to the worm propagation computation.

In the *Chain Binomial* model, an individual infected at time step t does not send out scans until the time step $(t+1)$. This approximation can bring concerns on the accuracy of the model when the worms use local preferential scanning, especially when the scanning rate is very high. For example, assume a time step size of 1 second, if a worm sends out 4000 scans per second, and has 10% preference of choosing the local /16 addresses, then within the same time step of infection, a newly

infected host can, on average, send out $4000 * 0.5 * 10\% = 200$ scanning packets to the local /16 network. Ignoring this can cause substantial underestimation of the worm propagation over the local network.

3.3 *Estimate Next Infection (ENI)* Model

When the *Chain Binomial* model is run stochastically, random numbers need to be sampled from a binomial distribution. Computing the inverse density function of Binomial is known to be a computationally expensive operation. The *Chain Binomial* model also assumes that the hosts infected within a time step cannot send out scans in the same time step. This assumption is problematic if the worm has a very high scanning rate and a strong preference for scanning local IP space. We developed Estimate Next Infection (*ENI*) model to cope with this problem.

3.3.1 *ENI* Model

In *ENI* model, for a given subnet j, at each starting point of the time step, two things are computed: i) new infections in this subnet that will happen in this time step; ii) scans that will be sent out to the other subnets in this coming time step.

$$r_j(t) = r_{jL}(t) + r_{jE}(t) \tag{3.12}$$

The received scanning rate r_j is treated as a summary of scanning rates from either *local source* r_{jL} or *external source* r_{jE}. r_{jE} is assumed to be constant during this time step. Initially the number of scans sent out from this subnet j in this time step is

$$s_j = I_j(t) * scan\_rate * time\_step \tag{3.13}$$

Assume that the received scans arrive at this subnet follow a poisson process, then the infection

that will happen also follows a poisson process with a rate of

$$\lambda_j(t) = r_j(t) * S_j(t)/C_j \tag{3.14}$$

where S_j is the number of susceptibles at that time and C_j is the CIDR block size of the subnet. Thus the next infection time can be sampled using an exponential process with the mean of $1/\lambda_j$.

$$\Delta t = exponential(\lambda_j(t)) \tag{3.15}$$

If the sampled time Δt is within this time step, the status of the subnet is updated to consider the scans sent out by this newly infected host from its infection time. Then r_L is updated, and the time is advanced to the sampled infection time.

$$r_{jL}(t + \Delta t) = r_{jL}(t) + \Delta r \tag{3.16}$$

$$s_j = s_j + \Delta s \tag{3.17}$$

where s_j is the number of scans that are to be sent out in the coming time step. The above process is iterated. The iteration ends when the estimated infection time falls out of this time step.

After this estimation process, the number of scans sent to other subnets in this time step $s_j(t)$ is computed and fluid flows are sent out. In the current implementation, fixed point computation is used to compute how those scans compete bandwidth and arrive at their destination subnets. At the destination, with parameter `scan_pkt_size` and `time_step`, a received scan rate can be easily computed.

We now talk about the computing of $r_{jE}(t)$, scan rate from external sources in equation 3.12. In the *ENI* model, at the beginning of each time step t, the scans sent out from subnet j needs to be computed and one important factor is the scans from external sources in this coming time step t. The exact $r_{jE}(t)$ is not available at the beginning of the time step, so approximation is used here.

We estimate its value from $r_{jE(t-1)}$ and $r_{jE}(t-2)$ by applying a simple interpolation as in (3.18).

$$r_{jE}(t) = r_{jE}(t-1) + (r_{jE}(t-1) - r_{jE}(t-2)) \tag{3.18}$$

3.3.2 Different Scanning Schemes

Local preferential scanning is often used by Internet worms. Having more preference to sending scanning packets to a destination with the same prefix can significantly speed up worm propagation within the local subnet, especially if it penetrated a firewall of this subnet. Studies show that a simple rule of having 20% preference of scanning local class B (/16) network tends to get a good performance for worms in most situations [52].

The impact of this can be incorporated into *ENI* model. Assume that there is a preference P_c of scanning local CIDR space C_L, and the *wormnet* j CIDR block size is C_{jL}, there are two possibilities depends on the relationship between C_L and C_{jL}.

- If $C_L \leq C_{jL}$, then all local preferential scans fall into the same *wormnet* j, thus the percentage of scans from the subnet j to itself is

$$P_L = P_c + (1 - P_c) * C_{jL}/C \tag{3.19}$$

  where C is the total CIDR space, i.e. total IP space.

- If $C_L > C_{jL}$, then only part of the local preferential scans fall into the same *wormnet*, we would have

$$P_L = P_c * (C_{jL}/C_L) \tag{3.20}$$

We can compute Δr in equation 3.16 using

$$\Delta r = scan\_rate * P_L. \tag{3.21}$$

Then scans from *wormnet* j to *wormnet* k consequently have two situations:

- If $C_L > C_{jL}$ and *wormnet* j and k are in the same C_L space,

$$s_{jk} = Binomial((s_j * (1 - P_L) + local\_scans), (C_k/C)) \tag{3.22}$$

- All other situations,

$$s_{jk} = Binomial(s_j * (1 - P_L), (C_k/C)) \tag{3.23}$$

In the implementation *poisson* inverse density function is used to approximate the *binomial* inverse density function because the probability C_k/C tends to be small in practice.

If more complicated local preferential scanning scheme is used, the computation of Δt and s_{jk} needs to be modified accordingly in the current implementation.

Sweeping is another scanning scheme that is less often used by Internet worms. An infected host would usually pick a starting point in the address space, and start scanning the address space following that point sequentially. If another already infected host is encountered, it picks another starting point for sweeping. *ENI* model cannot be directly applied to this kind of worms because it does not do random scanning.

3.3.3 Validation

In this section I present the experimental results to validate the *ENI* model. In the validation experiment we actually used optimizations and adjustments that we would discuss in section 3.5.

The validation is based on the Code Red v2 scenario. On the days of Code Red v2 worm incident in July, 2001, data trace has been collected over a $/16$ network at the *Chemical Abstract Service* (http://www.cas.org). The data trace has the source IP addresses of those scanning packets, which reflects the number of infections and the worm dynamics during the propagation. The data trace is processed and the worm dynamics over the global Internet can be inferred from the processed data.

Table 3.3: ISP Topology from Rocketfuel Project

ISP	AS number	#Wormnet	#links
Exodus	3967	244	785
Above.Net	6461	366	1334
Sprint	1239	516	2111
AT&T	7018	729	2984

In the experiment we used dataset from the *Rocketfuel* project at University of Washington [1]. Four ISP backbone topology in the United States are generated. *Wormnets* are attached to the routers of the backbone topology. We used the backbone topology of *Exodus* here in the validation. The other topologies are used in other experiments that we will discuss in other sections. Some information of the four ISPs are shown in Table 3.3

From the processed real-world data trace, we assumed that the total number of susceptibles is 374,500. There is no preferential scanning involved in Code Red v2 incident, and the scanning rate is set to be 5 scans/sec. Code Red v2 worms use TCP to convey the exploit, and a windows machine has a 21 second timeout time for TCP *SYN* packet. Although an infected host in Code Red v2 has 100 threads doing scanning concurrently, most scans hit unused addresses or hosts that do not have open HTTP services, which in turn causes a lot of timeouts, so the average scanning rate is around 5 scans/second.

We run the *ENI* model in deterministic mode and generated the worm propagation trace, and both the simulated trace and real-world data trace are plotted in Figure 3.2. When Code Red v1 was initially released, there was a bug in its random number generation scheme and it used a fixed seed. This bug was there for days before someone fixed it and re-injected the fixed code into the Internet. The fixed code was called Code Red v2, but that incident caused much longer delay in the starting

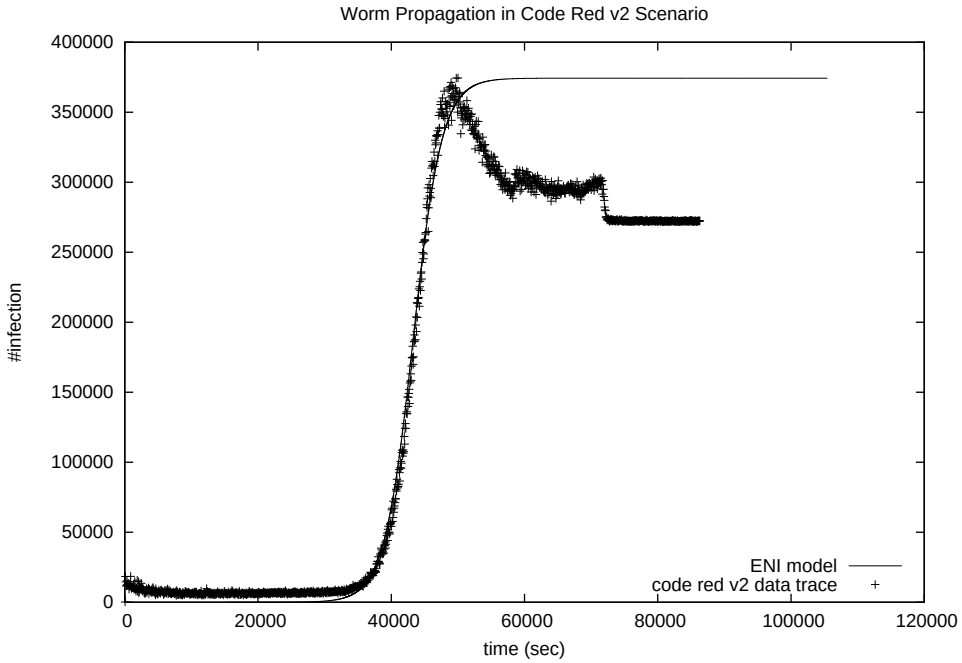

Figure 3.2: Validation using Code Red v2 trace.

phase of the worm propagation. Therefore, in the figure the starting point of the simulation trace is shifted to eliminate possible effect of that incident. As shown in the figure, the propagation phase of those two traces match fairly well. In the end, the number of infections in the real-world trace decreased, which was caused by multiple factors such as patching effort and device shutdown in some networks

To demonstrate the difference between the *ENI* model and the *Chain Binomial* model, an experiment has been conducted. The experiment used the *Sprint* backbone topology with 566 *wormnets*. The total number of susceptibles is 75,000, scanning rate is set to 4,000 scans/sec. In both runs, there is a 25% preference of scanning local /16 networks. In *Chain Binomial* model, it is assumed that for a given *wormnet* j and a time step t, the scans from all sources are in constant rate; while in *ENI* model, it accounted for the scans from newly infections in *wormnet* j within this time step t. Given 25% preference of scanning the local addresses, the effect of those scans should not be

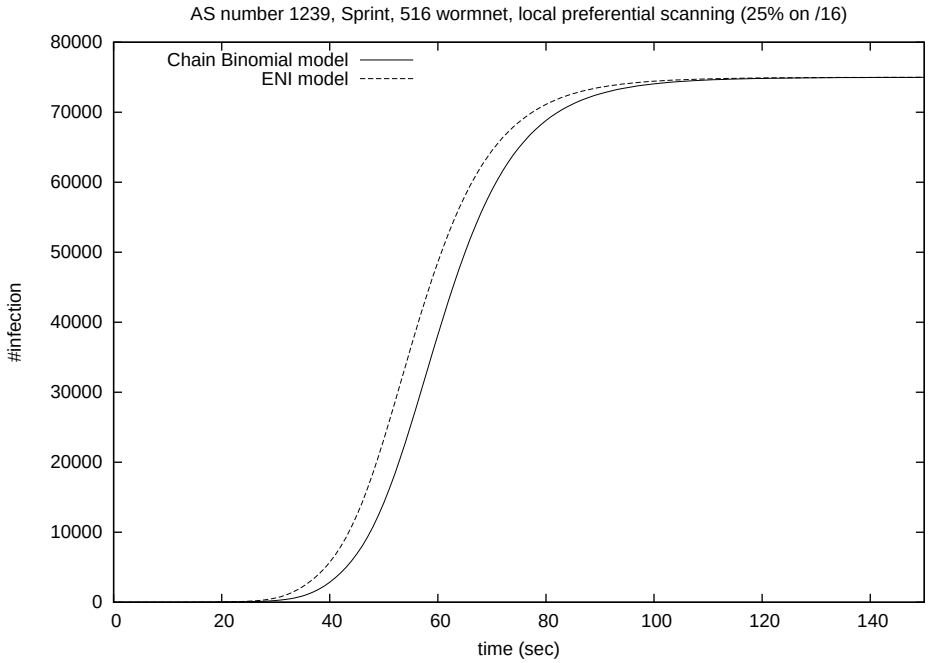

Figure 3.3: *ENI* model v.s. *Chain Binomial* model

simply ignored. As shown in Figure 3.3, there is a noticeable difference between the dynamics of the two models.

3.4 Multi-Resolutions in Modeling and Simulating Internet Worms

3.4.1 Resolutions in Modeling and Simulation

When simulating a complicate system, it is almost certain that some trade-off exists between the level of simulated details and the required computing resources. Intuitively the more details a simulation exhibits, the more computing resources it consumes. When it comes to the modeling and simulation of Internet worms, resolutions in different dimensions need to be considered. They have significant influences on the accuracy, the performance, and possible usage of the models.

Traffic Representation In network simulation, the representation of the traffic usually has a large

impact on the simulation. One can choose to represent each single packet using an *event*, thus the term *packet-oriented simulation*. This is a natural choice of simulating a network in fairly detail. However, if the simulated network topology is large, usually a large amount of traffic is expected. The simulation would then generate many events, and become computing resource intensive.

From the experience, people find that sometime a small portion of the traffic is the focus of the simulation, and sometime the contents of the traffic do not need to be elaborated in the model. Therefore, models have been created in which a train of packets is represented by a *fluid flow* [23, 24, 30, 31, 40, 47]. In a normal *fluid-oriented* model, an *event* will be generated when there is a change in the rate of the traffic fluid, and usually this can significantly reduce the number of *events* generated in a simulation. This rate-change *event* is propagated gradually in the simulated network, and may in turn generate more simulation *events* due to congestion [31, 40].

To avoid the possible event explosion situation in the *fluid-oriented simulation*, various techniques can be used, and one of them is to use a more crude level abstraction. In this abstraction, the transit states of the fluid rate change are ignored, i.e., it always tries to compute the steady state of the traffic in the network [5, 41]. This approach is called *fixed-point computation* in fluid-oriented simulation.

Time In a fluid-oriented Internet worm model, time resolution has become an option as well. In such a model, the time has been divided into discrete slices. The granularity of the time slice can affect the accuracy of the model as well as performance. The choice here is straightforward. The finer the time slice is, the more computation is required, but the more accurate the simulated result might be. In a more general case, when simulating slowly evolving systems, one can choose larger time slices; a finer time slice is preferred if the states of the simulated system change very quickly or frequently. When different parts of the system evolve at different speeds, applying individual time resolutions on different parts might be considered.

Topology Representation The Internet now has millions of computing devices. In the simulation it is very difficult to simulate as many devices with the current technology. Simulating computer worms on top of a much smaller network topology has some implications on the simulated results and can bring artificial effects [59]. A more common approach is to provide another level of abstraction in the network topology. In this abstraction, a node in the simulated topology might be used to represent a subnet. It keeps some state information of the simulated devices inside the subnet but omit all detailed internal topology details. The size of this represented subnet depends on the granularity of abstraction.

3.4.2 Combine Multiple Resolutions

We presented *ENI* model in section 3.3. In fact, the *ENI* is a model with multi-resolution in time. When we consider the scanning traffic received by *wormnet* j, two parts of the traffic are considered: scans from sources inside the same *wormnet* $r_{jL}(t)$ and scans from external sources $r_{jE}(t)$. For $r_{jE}(t)$, scans from external sources, we considered it to be constant within a time step t. In another word, the resolution on the external scans is the `time_step` size. This approximation is inspired from the following observations: the CIDR block size of *wormnet* j is relatively small compared with the total IP space (true in most cases). Given random scanning, only a small portion of the scans fall into *wormnet* j. Ignoring this in a short period of time is an acceptable approximation. On the other hand, for $r_{jL}(t)$, the resolution is smaller than `time_step` size because every time we advance Δt inside a time step we update the $r_{jL}(t)$ to $r_{jL}(t+\Delta t)$. This finer granularity is because if there is local preferential scanning, a new infection inside the local networks can potentially increase the scanning over the local network significantly even within the same time step, especially if the worm has a significant preference for local IP addresses as suggested in [52].

Having a multi-resolution approach on traffic representation and topology can be an approach that helps scalability while preserver detailed packet level information. As shown in Figure 3.4, a backbone is formed using routers and *wormnets*. Fluid-oriented worm models can be applied on

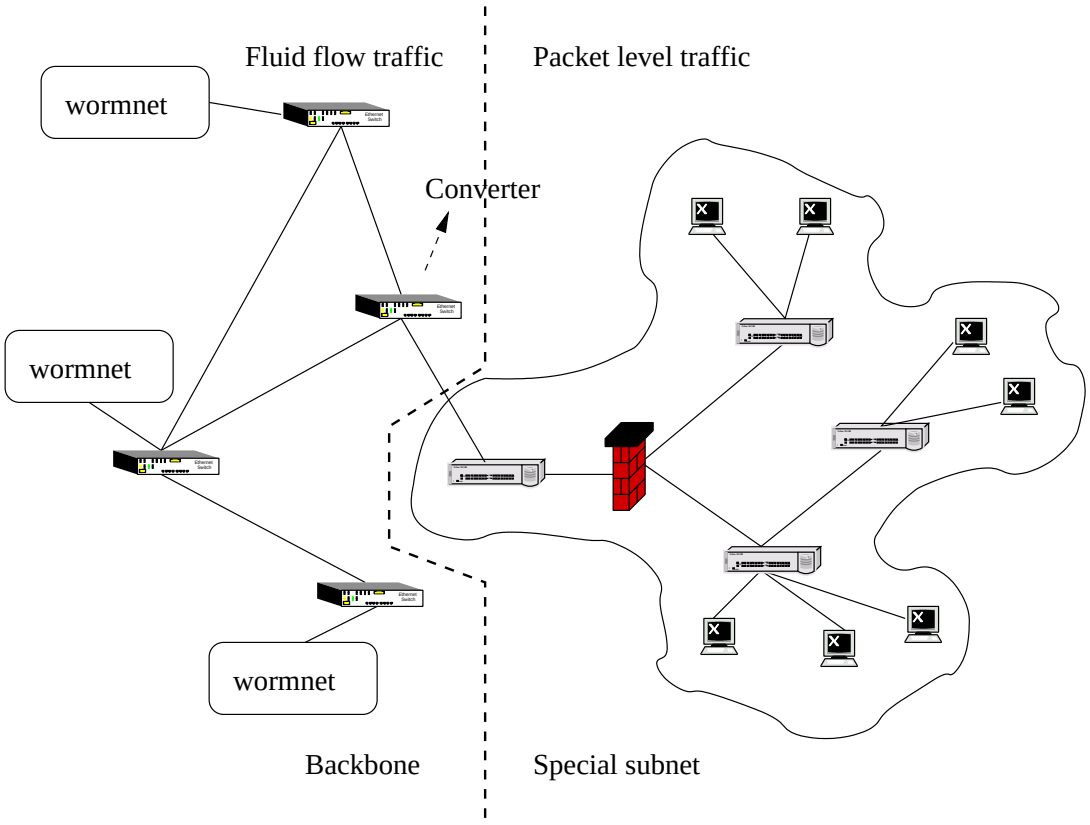

Figure 3.4: Multi-Resolution in topology and traffic

top of it. For a particular *wormnet*, a traffic converter is implanted, and the converter connects to a subnet with detailed internal topology. Within this special subnet, the worm traffic from the outside is converted into discrete packets then sent in. The worm scanning packets that go out will be again intercepted by the converter and fluidized into fluid flows. By using the abstracted backbone topology and fluid flow scanning traffic in large portion of the network model, the scalability advantage can be kept. By preserving packet level traffic inside the special network, some techniques that require packet level information can be exercised and studied.

3.4.3 Defluidize and Fluidize Scanning Traffic

The key component of implementing the multi-resolution model is the traffic converter. The main functionality of the converter is to defluidize and fluidize worm scanning traffic. In the current implementation, worm scans based on TCP are not considered in the defluidizing and fluidizing. The TCP is connection-oriented, and it applies congestion avoidance algorithm to adjust its transmission speed. It is not clear yet on how to convert TCP packets of different connections into aggregated fluid TCP flow.

Fluidizing the scanning traffic is straightforward. The converter intercepts the outgoing worm scanning packets, accounts them based on which *wormnet* their destination IP belongs to, and sends out aggregated fluid flows every time step.

Defluidizing is comparatively more sophisticated. Let us assume that the converter is just on *wormnet* j. In the current implementation, the flow merging optimization discussed in section 3.5.2 cannot be applied to fluid flows that go to this *wormnet* j any more. (The optimization can still be applied to the traffic go to other *wormnets*.) At each time step t, we have received scanning rate $r_{ji}(t)$. The packets from *wormnet* i are assumed to arrive with poisson process, thus the inter-arrival time can be sampled using exponential process with mean of $1/r_{ji}(t)$. In another word, we would probably have $N-1$ poisson processes on *wormnet* j, where N is the total number of *wormnets*.

When a scanning packet from *wormnet* i is generated, the critical fields that need to be filled

include `source_ip`, `dest_ip`, `source_port`, `dest_port`, and `time_to_live` (ttl).

- The fields `dest_ip` and `dest_port` are easy. The destination port number is a configured parameter and can be directly used. The destination IP can be randomly chosen from the CIDR block of *wormnet* j.

- In order to fill the other information, for each *wormnet* i, *wormnet* j needs to keep an array of *(*IP, port, ttl) tuples. Whenever a new infection happens in *wormnet* i, one more element needs to be added to the corresponding array. The tuple keeps information of the IP address, source port and TTL information for each "infected" host in *wormnet* i. The source IP and source port can be randomly generated as long as the IP falls in the CIDR block of *wormnet* i. The TTL can use the value from the fluid flow packet, minus some random value to account for the "internal topology" of *wormnet* i. The reason to keep all the detailed information is to keep the consistency of generated worm scanning packets.

3.4.4 Validation Experiments

To validate the correctness of the multi-resolution approach, an experiment has been set up. In the experiment, the "speical subnet" is a /16 network. Its detailed topology includes more than 300 hosts and routers, 140 of which are vulnerable hosts. We used *Brite* to generate a simple topology to be used as the backbone, and the backbone has just 32 *wormnets*. 75000 vulnerable hosts are populated inside the backbone part of the topology, and one of them is initially infected. In the experiment, only uniform scanning is used, i.e., there is no local preferential scanning. We focus on the worm propagation dynamics inside the special /16 subnet, which can be indicated by the data below.

- The number of received worm scans from the outside to this special wormnet. This is the main driving force for the worm spread dynamics inside the special wormnet.

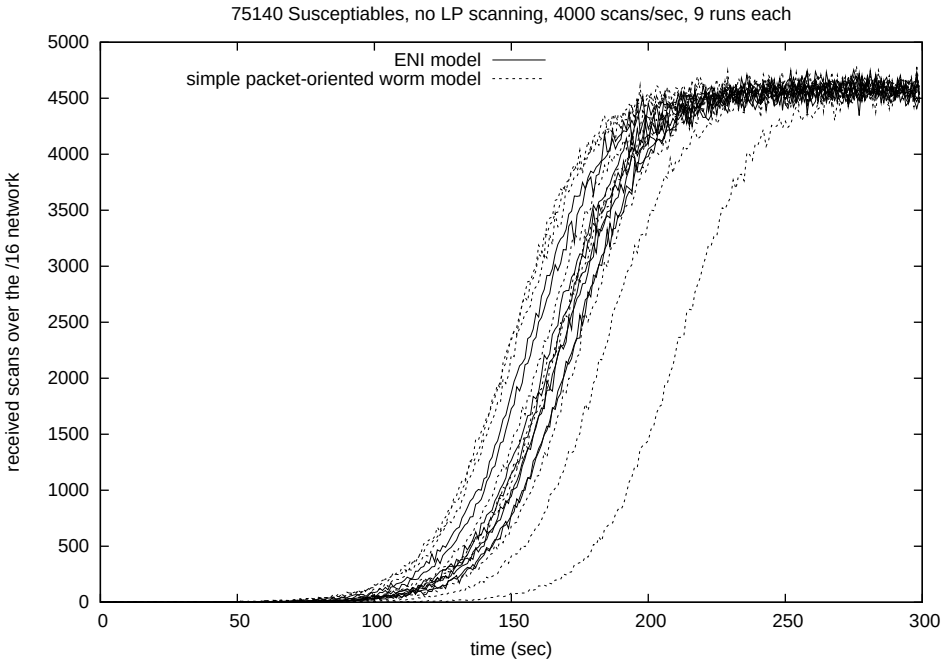

Figure 3.5: Received scans over the /16 subnet

- The histogram of the number of infections inside the special subnet. This directly reflects the worm dynamics inside.

Simple Packet-Oriented Worm Model

We need some baseline results with which we can compare the results from the multi-resolution experiment runs, and we developed a very crude packet-oriented worm model for this purpose.

Inside this simple packet-oriented model, we do not have any topology. We have an array of n hosts, each can be *susceptible* or *infected*. Given that $I(t)$ hosts have been infected at time t, the next time a scan will happen can be computed using $\Delta t = 1/(scanrate * I(t))$. We can then advance the simulation time of this model to $(t + \Delta t)$, randomly sample an integer k from $[0, 2^{32}]$. If k is inside $[0, n)$, then the host with index k becomes *infected* if it is still *susceptible*. Through an iterating process, we can simulate the uniformly scanning worm dynamics in the high level in

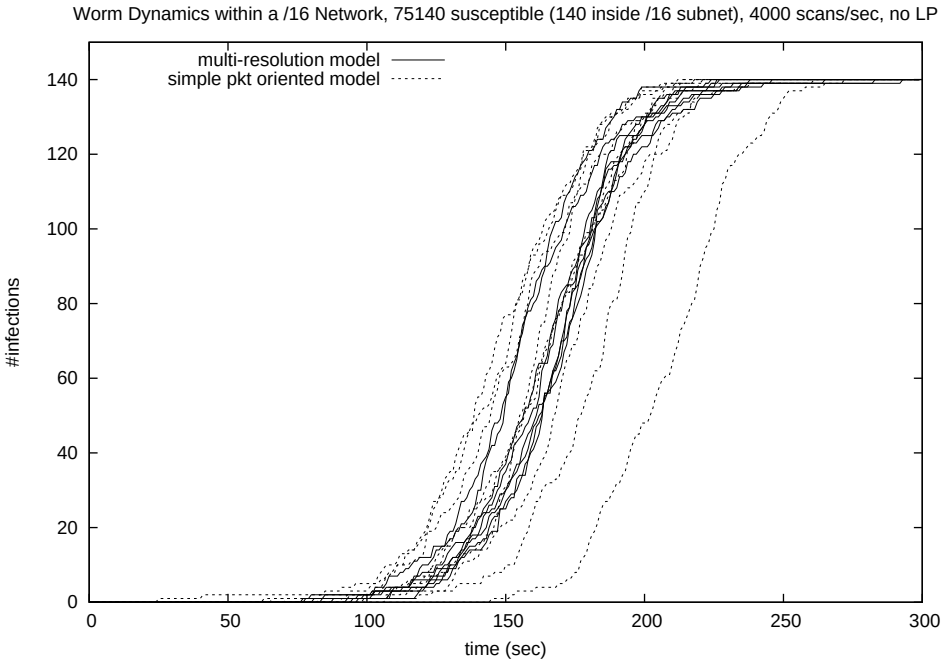

Figure 3.6: Worm popagation dynamics inside the /16 subnet

a packet-oriented fashion. To collect the same data as in the multi-resolution approach, we can focus on the number of scans fall in the scope $[(n - n'), (n - n' + C')]$, where n' is the number of susceptible hosts inside the special subnet, and C' is the CIDR block size of the speical subnet. We can also collect data on how worms propagate among hosts of range $[(n - n'), n]$.

Experiment Results

We run both the simple packet-oriented model and the multi-resolution model in stochastic mode, and plotted the results in Figure 3.5 and Figure 3.6. Figure 3.5 displays the number of received scans inside the special /16 network, and Figure 3.6 displays the worm spreading process inside the /16 network. From the results we can see that the worm propagation dynamics in both model exhibit the same characteristics and trend. It is noted that in simple packet-oriented model, the results seem to have more stochastic effect. We need some further investiagtion for the exact cause of such effect.

3.5 Other Pieces of the Puzzle

Having the worm propagation model alone does not complete all the pieces needed for worm simulation. This section discuss the optimizations and other problems need to be solved in order to improve the performance and conduct the simulation.

3.5.1 Routing and Forwarding

Routing is one important aspect of network simulation. When exercising random scanning worm scenarios, the scanning traffics are generated from all sources to all destinations. If on-demand route computation [27] is used, special caching can significantly reduce total runtime. If packet-oriented model is used, many sources/destinations would be in the same subnets. Therefore appropriate route aggregation, if supported, can reduce the memory usage significantly by shrinking the route forwarding table , and can consequently improve the runtime. I will not discuss the above in this thesis but will focus on the interactions between the routing scheme and worm propagation.

In a real-world worm incident, when the scanning packets are sent, roughly three situations may happen:

1. The destination IP address belongs to an unadvertised CIDR block. Because local hosts and routers do not have the full routing map, the packet will be delivered using a *default route*. It should be eventually dropped at the first router that does have a full routing map, i.e., a BGP router that does not have *default route*. Usually it is the first main access router of the ISP, right before the packet can reach the backbone. *Default route* is the route used to forward packets when no other route is available.

2. The destination IP address belongs to an existing host. There is no problem in this scenario, the packets should normally be delivered.

3. The destination IP address belongs to an advertised CIDR block, but it is not used by any

host or router. The packet should at least be delivered to one gateway router of the subnet that owns that CIDR block, and dropped by some router inside that subnet when more detailed information is available.

In a packet-oriented simulation, situation 2 is naturally covered. For packets that go to unadvertised space, they are taken care of as long as *default route* is supported. One needs to be careful with packets that need to be delivered to an unused address in advertised CIDR block. If in the model both inter-domain (BGP) and intro-domain (OSPF, etc.) routing are used, the hierarchy often ensures that the packet can be delivered to a router in the destination CIDR block before being dropped. If hierarchical routing is not supported, very likely the source host of a scan packet will immediately discover that there is no route for that particular packet and drop it too early.

If fluid-oriented worm model is used, the situation is slightly different. Situation 2 is naturally covered. For packets that go to unused hosts inside an advertised space, the packets are counted inside the traffic flow and will be delivered to the correct subnet. However, for traffic that are destinated to unadvertised space, the system would not able to setup paths for those scanning fluid flows because there is not route. In current iSSFNet implementation we incorporate this inside the worm model. Assume that each time step the total number of sent out scans is s_j, the scans go to *wormnet* i is s_{ji}. We then have

$$s_d = s_j - \sum_{i \neq j} s_{ji} \tag{3.24}$$

where s_d is the rate of a traffic flow that should be sent to unadvertised space. Such a flow is computed at each *wormnet* in every time step and sent to the first router that has a full BGP map.

3.5.2 Optimizations in Fluid-Oriented Worm Model

Flow Merging

In the fluid-oriented models, each *wormnet* sends out flows to every other *wormnets*. If the number of *wormnets* is N, then the total number of flows is N^2. The time complexity of this is difficult

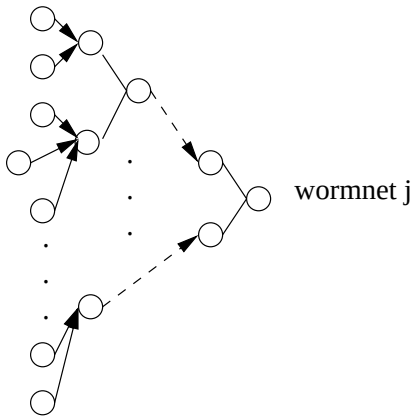

Figure 3.7: Analysis of space complexity after applying flow merging optimization

to estimate because it depends very much on the topology and whether there is any loop of traffic congestion. The space complexity of this is $O(N^2H)$, where H is the average number of hops a flow traverses.

Two properties motivate an practical optimization:

- For the purpose of computing worm propagation, the detailed contents of each single scanning packet are insignificant. The simple fact of receiving a scanning packet at a vulnerable host is enough to determine an infection. That is also the exact reason we can use fluid-oriented model for worm propagation.
- There are N^2 flows, while only N possible destinations.

Given the two properties, an intuitively simple optimization is to merge the flows that have the same destinations.

After applying this optimization, the space complexity become $O(N^2)$. As shown in Figure 3.7, if we only consider the traffic flows that have destination as *wormnet* j, regardless of the topology and the routing schemes or policies, the traffic flows will eventually form a tree pattern. Given N *wormnets*, we have $N-1$ edges in the tree. Since we apply flow merging, there would be at most one flow in each link, so the space cost of all flows that go to *wormnet* j is $N-1$. Because we have

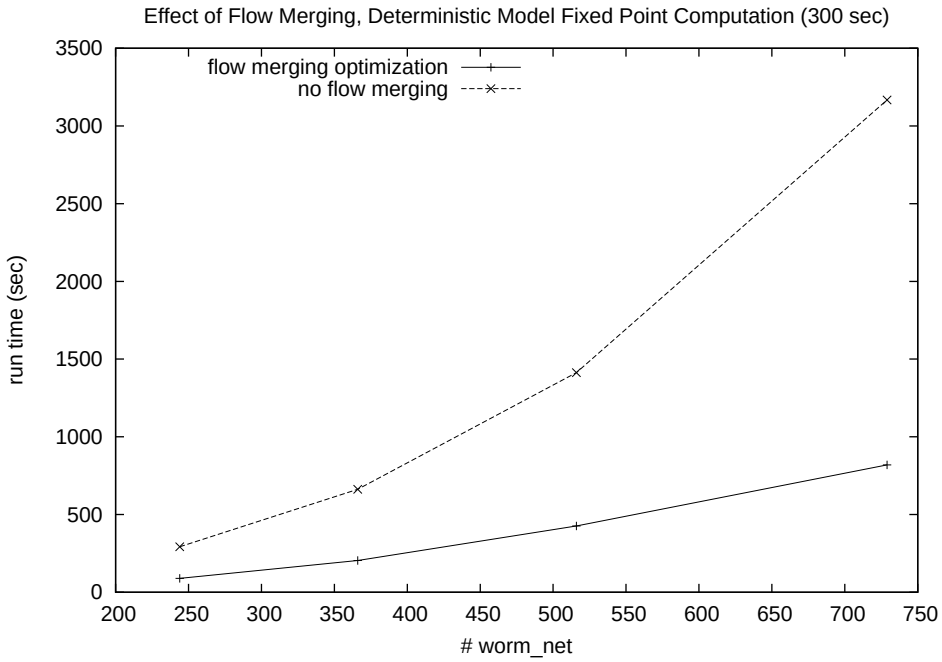

Figure 3.8: Flow-merging effect on simulation runtime

N *wormnets*, so the total space complexity is $O(N^2)$.

The time complexity, again, is difficult to compute analytically. However, empirical results show that in a fairly realistic topology, the optimization can significantly reduce run time. We again used the topologies generated from *Rocketfuel* project. Some details of the topologies can be found in Table 3.3.

In the Internet worm scenario exercised, there are 75,000 vulnerable hosts, the advertised space is kept to be 25%, the worm scanning rate is 4,000 scans/sec, and there is a 25% local scanning preference on local /16 networks. Two groups of experiments were run deterministically, and the simulation run time is collected and plotted in Figure 3.8. It is clear that *flow merging* benefits the simulation runtime, especially in larger topologies.

Deterministic v.s. Stochastic

In the *ENI* model, there are several places we need to generate random numbers from certain distributions. The model can run both deterministically and stochastically. Intuitively running stochastically costs more time because we need to generate random numbers, but it brings in stochastic effects that are desirable in some situations.

Experiments show that we can use a hybrid running mode to have stochastic effects while improve the performance in the same time. It is motivated by the fact that the randomness in the starting phase of the worm propagation is often the most important part that determines how fast the worm can propagate. After a significant portion of the vulnerable hosts are infected, the number of scans is often large enough so that running the simulation in deterministic mode does not make much difference in terms of propagation. So the approach is to run the model stochastically in the beginning, but once the number of infectives exceeds a threshold, switch to deterministic mode. The threshold can be chosen empirically.

To demonstrate its effect, we have three groups of experiments. They are based again on the Rocketfuel dataset. The three groups are run deterministically, stochastically and in hybrid mode respectively. In Figure 3.9 we have the comparison of the stochastic model and hybrid model on Exdous backbone network. Ten runs are conducted on each mode, and one can see that the dynamics of worm propagation is preserved nicely in the hybrid model. When the hybrid model is used, intuitively the runtime is better than the runtime of using stochastic model. In fact, it actually performs better than the deterministic model as well. This is because that in the starting phase there are less traffic flows when the hybrid model is used. In Figure 3.10 we plotted the runtime of the experiments on the four ISP backbone networks, and this effect is fairly obvious .

The confidence interval of the stochastic runs on Above.net is relatively large, further investigation confirmed that it was caused by the susceptible hosts distribution in the setup. Given 75,000 vulnerable hosts and 366 *wormnets*, we distributed the vulnerable hosts to the *wormnets* using an exponential distribution. The particular distribution for Above.net in the experiment happens to

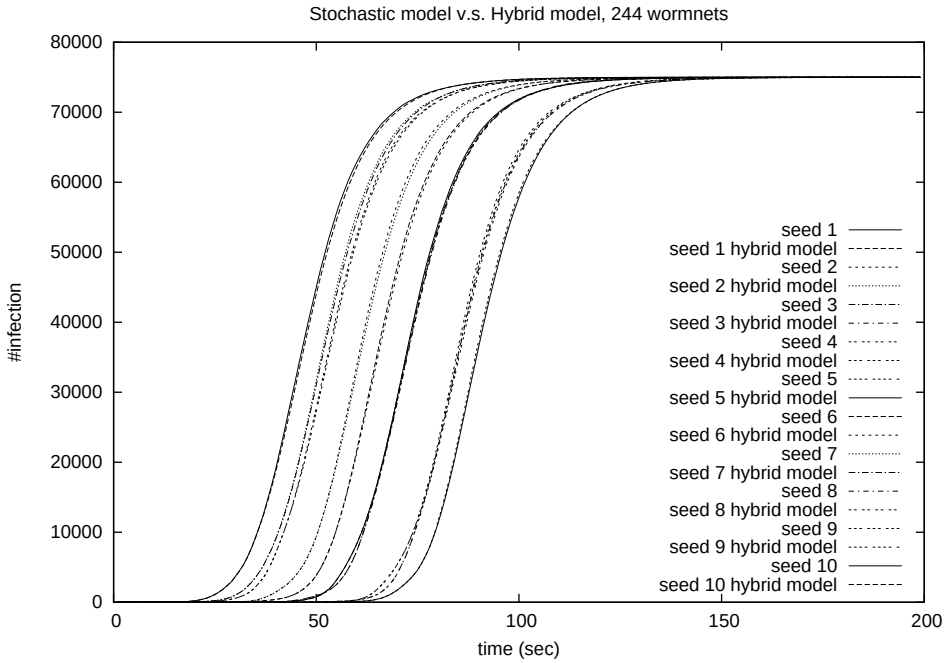

Figure 3.9: Comparison of stochastic mode and hybrid mode

be more uneven when compared with the other three ISP networks. When there is a significant preference to scanning local addresses, this unevenness caused more variability in stochastic runs. Some further experiments revealed some interesting rule of worm propagation and is discussed in section 3.5.3

3.5.3 Other Experiments

There are some other experiments and results that cannot be easily included in the previous sections, and they are organized into this section.

Impact of Varying Time-Step Size

One parameter in the fluid-oriented worm model is the `time_step` size. Given a fixed simulation time, intuitively the runtime cost is linear to the number of time steps, thus inversely proportional

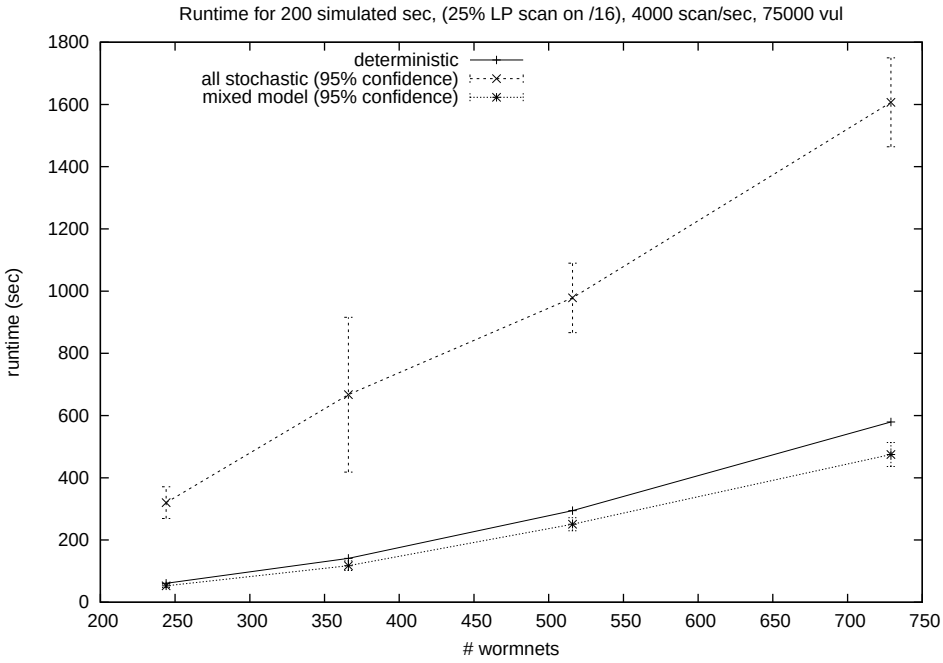

Figure 3.10: Runtime of deterministic mode, stochastic mode and hybrid mode.

to the `time_step` size. From the modeling point of view, a too large `time_step` size may introduce errors into the worm simulation results, especially when the worm spread is fast. Other other hand, a `time_step` size that is smaller than the average packet traverse time will also have negative impact on the accuracy of the results. Ideally, the `time_step` size can set to be the expected time when an infection will occur in the system, but it is not a constant value in the worm propagation process. We choosed a simpler approach in the current implementation and used a fixed value.

Figure 3.11 has the experiment result of varying `time_step` size in a worm scenario with $75,000$ susceptibles, $4,000/sec$ scanning rate, As shown in the figure, even for very fast scanning worms, choosing a `time_step` size of 1 second does provide s fairly accurate result already, using $1/2$ second as `time_step` size would probably suffice for higher requirements for accuracy. In the other experiments of the thesis, the results are all based on 1 second `time_step` size.

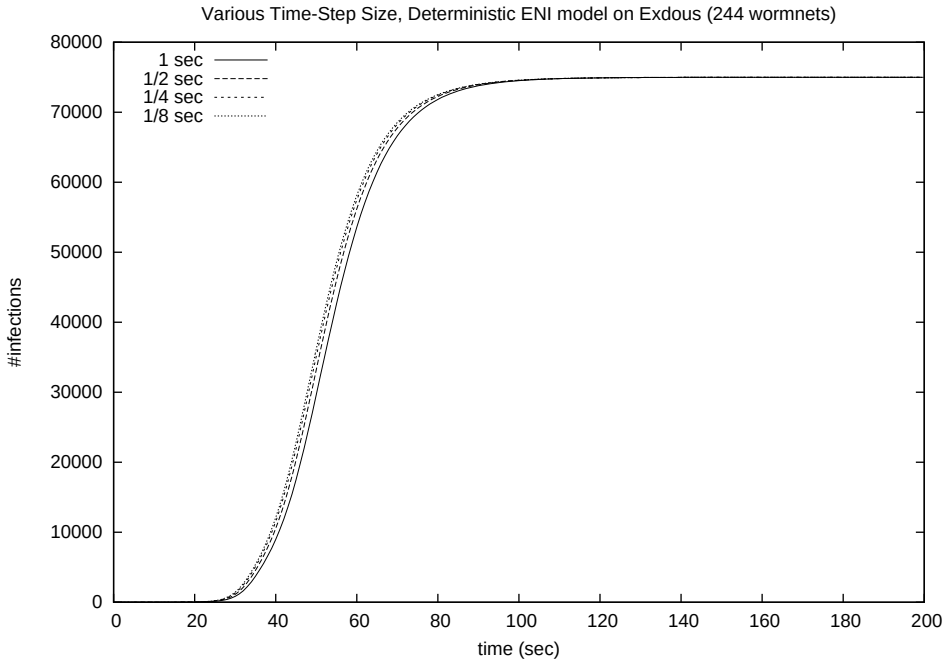

Figure 3.11: Impact of varying time-Step size

Local Preferential Scanning and Susceptibles Distribution

In this part we presents a group of experiment results that reveals an interesting connection between local preferential scanning and susceptible distribution. Simulation results show that *when there is a significant preference to scanning local subnet addresses, the more uneven the susceptibles are distributed in the network, the faster it tends to propagate.*

Although this rule is not very explicit from the first glance, it is not counter-intuitive. When the susceptibles are not evenly distributed, there are some clusters of susceptibles with higher density, local preferential scanning can take advantage of the higher density in those clusters and speed up the whole propagation. Notice that for the same reason, whether the initial infection happens in those larger clusters also has an impact on the simulation results.

We demonstrate this effect through a group of experiments. We used *Brite* [35] to generate a simple topology and attached just 32 *wormnets* to it. In all scenarios we use $80,000$ susceptibles,

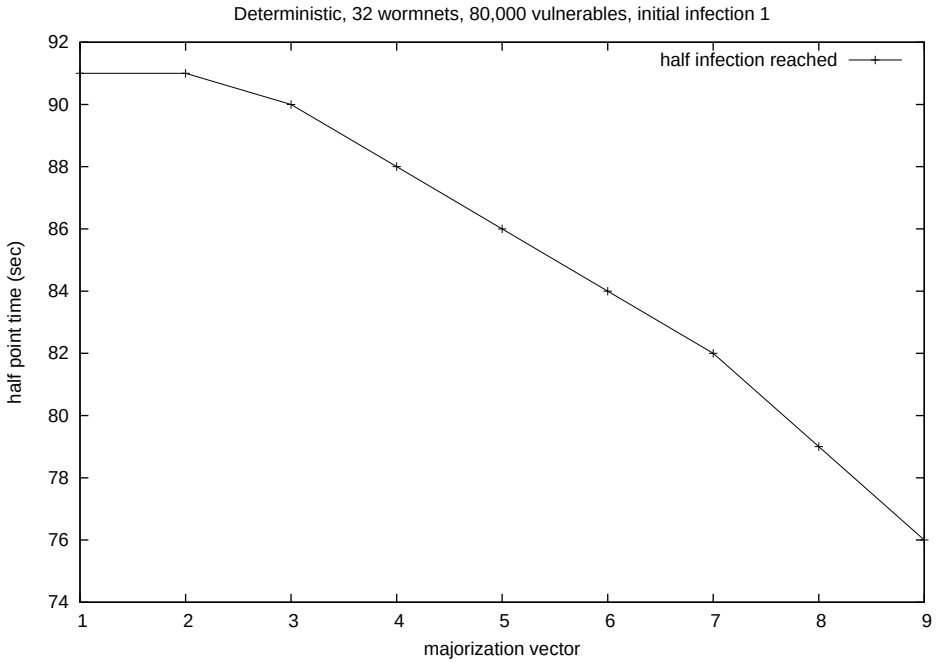

Figure 3.12: Impact of susceptibles distribution on Internet worm dynamics

$4,000$ scan/sec, 25% local preferential scanning, and the *wormnet* CIDR block size is chosen to be $/7$.

To generate different unevenness in susceptibles distribution, we use the concept of *majorization*. For vector $X = (x_1, x_2...x_n)$ and $Y = (y_1, y_2...y_n)$, given that $\sum_{i=0}^{n} x_i = \sum_{i=0}^{n} y_i$, It is said that X majorizes Y ($X > Y$) if

$$\sum_{i=1}^{j} x_{[i]} \geq \sum_{i=1}^{j} y_{[i]} \tag{3.25}$$

for all $j = 1, 2, ...n$, where $x_{[i]}$ denotes the i^{th} largest element in X. Now if we have two susceptibles distributions $D_1 = (a_0, a_1, ...a_{31})$, $D_2 = (b_0, b_1, ...b_{31})$ and $D_2 > D_1$, it is clear that D_1 is a more even distribution than D_2.

An array of distributions are created: D_1, D_2, ... D_9. The first element of all D_i here is always 2500 and has the initial infection. The array of distributions also has the property that $D_{i+1} > D_i$. In another word, in D_1 the susceptibles are most evenly distributed, while in D_9 they are the least

evenly distributed.

We run the simulation on the array of distributions deterministically and marked the time at which half of the susceptibles are infected and plotted the results in Figure 3.12. The x-axis is the index of that distribution in the array, i.e. the first point from left is for D_1, while the last is for D_9. The y-axis is the time at which half of the susceptible hosts have been infected. We can see that the more uneven the susceptible hosts are initially populated, the worm propagates faster in the system.

For a given distribution, the location of the initial infection also affects the worm propagation speed. We used D_3, D_6 and D_9 distributions, all but the first element inside each vector are sorted in increasing order. In another word, when $i > 1$, moving initial infection from *wormnet* i to *wormnet* $(i + 1)$ is moving it to a larger concentration of susceptibles. Figure 3.13 plots the results of varying the location of the initial infection. As we expected, when the initial infection move to larger concentrations the propagation become faster. This effect is more prominent if the distribution itself is more uneven.

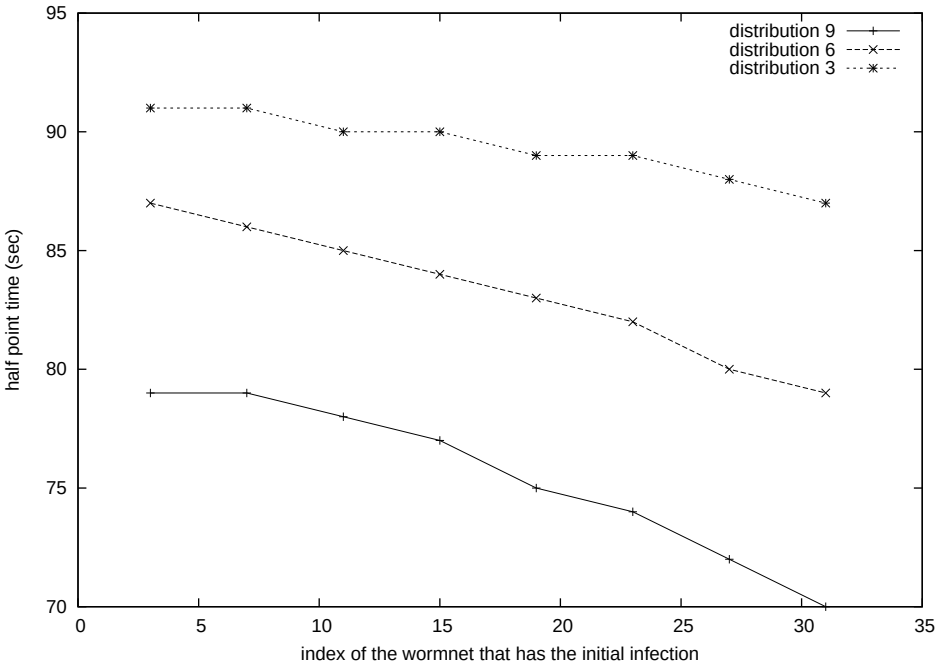

Figure 3.13: Impact of varying initial infection location on Internet worm dynamics

Chapter 4

Summary and Future Work

4.1 Summary

This thesis presents the research work of building an immersive simulation environment for security related studies, and in particular, presents the case study of using the environment for Internet worm modeling and simulation. The thesis can be summarized as below:

- *Real-time Immersive Network Simulation Environment (RINSE)* has been built as an effort to provide automated tools for security related simulations and studies. One major component of *RINSE* is a network simulator: *iSSFNet*. *iSSFNet* is built on top of iSSF, a high-performance conservative discrete event simulation engine.
  - *iSSFNet* facilitates the design and implementation of protocol models. It supports dynamic configuration of network models running various networking protocols. It allows a mixture of fluid-oriented model and packet-oriented model coexist in the same network model.
  - Taking advantage of the lower layer simulation engine, *iSSFNet* supports parallel execution both with shared-memory multiprocessors and in distributed environment. The

infrastructure design of *iSSFNet* also provides a layer of abstraction to hide details of parallel execution from users and developers of networking protocols.

  - *iSSFNet* supports human-in-the-loop interaction during runtime. A centralized user command distribution infrastructure is implemented inside *iSSFNet* to support efficient command processing. A dozen of commonly used user commands have been implemented and it is easy to add in new commands in the future.

- One component of the research on the defense against Internet worms is to have an environment that can be used to study worm behavior and exercise worm related incidents. The modeling and simulation of Internet scanning worms is studied using *iSSFNet*.

  - The *Estimate Next Infection (ENI)* model has been developed in *iSSFNet*. The model incorporates important factors such as network topology and traffic congestion in computing worm dynamics. It is also a model that exercise multi-resolution in time domain to compensate for worms that have a significant preference to scanning local networks.

  - Resolutions in different domains on worm modeling are discussed. We also presented the work to combine worm models with different resolutions in topology and traffic representation, preliminary results are reported.

  - Some optimizations in fluid-oriented work simulation have been applied to the worm model in *iSSFNet*. The effect and impact on the performance are tested through experiments. A hybrid approach is proposed so that one can keep the stochastic effect in the worm simulation while slightly improving performance in the same time.

  - Some other worm related experiment results are reported. A connection between local preferential scanning and initial susceptibles distribution is revealed and demonstrated using experiments.

4.2 Future Work

There are several tasks that seem to be natural follow-ups for the current work I am doing, and can be described as possible short term future work:

- Some further optimizations can be applied to fluid-oriented Internet worm models. To be more specific, one approach is *flow splitting*. Instead of sending out multiple flows from the *wormnets*, sending out an aggregated scanning traffic flow, split the flow when it traverses the network. This I believe is going to further improve the simulation performance in practice.

- The defluidize / fluidize converter has some preliminary results, but there are other possible implementations, a deeper study may bring some further understanding of the general problem of combining fluid-oriented traffic with packet-oriented traffic.

- The current worm model in *iSSFNet* has not taken the advantage of parallel execution yet. The implementation is generally straightforward except some technical problems that may be caused in defluidizing worm scanning traffic. A parallel version of the worm model can run on a larger topology that we cannot afford right now with sequential execution.

For relatively long term, there are open problems that I am interested in and may become future direction. I am still interested in the security related simulation problems. For Internet worms, it has almost come to an agreement that automated system is necessary to help worm detection and defense, but building such system still involve many open questions. For example, if there is an alarm, assume that it carries a proof of its validity, how can it be efficiently distributed in large scale? This particular problem may touch fields such as peer-to-peer networks and overlay networks, and a simulation environment is exactly what is needed for designing and exercising. I have also conducted research on wireless ad-hoc networks. I am in general interested in the wireless ad-hoc routing and the simulation of wireless ad-hoc networks. The wireless network simulation is generally more computation intensive than simulation for wired networks. The sophisticated lower

layer interactions and node movement make it difficult to parallelize, but a good solution to this is also attractive.

Bibliography

[1] T. Anderson, R. Mahajan, N. Spring, and D. Wetherall. Rocketfuel: An isp topology mapping engine. http://www.cs.washington.edu /research/networking/rocketfuel/, 2005.

[2] J. Banks, J. S. II Sarson, B. L. Nelson, and D. M. Nicol. *Discrete-Event System Simulation*. Prentice-Hall international series in industrial and system engineering. 3rd edition, 2000.

[3] V. H. Berk, R. S. Gray, and G. Bakos. Using sensor networks and data fusion for early detection of active worms. In *Proceedings of the SPIE Aerosense conference*, Orlando, FL, April 2003.

[4] R. Bradford, R. Simmonds, and B. Unger. A parallel discrete event ip network emulator. In *Proceedings of the 9th International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'00)*, 2000.

[5] T. Bu and D. Towsley. Fixed point approximations for tcp behavior in an aqm network. In *Proceedings of ACM SIGMETRICS 2001*, June 2001.

[6] Cert advisory ca-2001-26 nimda worm. http://www.cert.org/advisories/CA-2001-26.html, 2001.

[7] X. Chen and J. Heidemann. Detecting early worm propagation through packet matching. Technical report, University of Southern California, Information Sciences Institute, Februray 2004.

[8] Z. Chen, L. Gao, and K. Kwiat. Modeling the spread of active worms. In *Proceedings of IEEE Infocom 2003*, 2003.

[9] D. M. Cowie, J. H.and Nicol and A. T. Ogielski. Modeling the global internet. *Computing in Science & Engineering*, pages 30–38, January - February 1999.

[10] J. H. Cowie. Parallel discrete-event simulation in java. In *Proceedings of ACM 1998 Workshop on Java for High Performance Netwrok Computing*, 1998.

[11] J. H. Cowie, H. Liu, J. Liu, D. M. Nicol, and A. T. Ogielski. Towards realistic million-node internet simulation. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 1999.

[12] J. H. Cowie, D. M. Nicol, and A. T. Ogielski. Modeling 100,000 nodes and beyond: Self-validating design. In *DARPA/NIST Workshop on Validation of Large Scale Network Simulation Models*, May 1999.

[13] D. J. Daley and J. Gani. *Epidemic Modelling: An Introduction*. Cambridge University Press, 2001.

[14] S. Das, R. F. Fujimoto, K. Panesar, D. Allison, and M. Hybinette. Gtw: A time warp system for shared memory multiprocessors. In *Proceedings of the 1994 Winter Simulation Conference (WSC'94)*, pages 1332–1339, December 1994.

[15] M. W. Eichin and J. A. Rochlis. An analysis of the internet virus of november 1988. In *IEEE Symposium on Researchiin Security and Privacy*, 1989.

[16] K. Fall. Network emulation in the vint/ns simulator. In *Proceedings of the Fourth IEEE Symposium on Computers and Communications (ISCC'99)*, July 1999.

[17] S. Floyd and V. Paxson. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking*, 9(4):392–403, August 2001.

[18] Wormscout anti-worm solution components. Forescout, http://www.forescout.com/ worm-scout.html.

[19] R. M. Fujimoto, K. Perumalla, A. Park, H. Wu, M. Ammar, and G. F. Riley. Large-scale network simulation -how big? how fast? In *Symposium on Modeling, Analysis and Simulation of Computer Telecommunication Systems (MASCOTS)*, October 2003.

[20] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *IEEE Symposium on Security and Privacy*, 2004. to appear.

[21] J. O. Kephart, D. M. Chess, and S. R. White. Computers and epidemiology. In *Proceedings of IEEE SPECTRUM*, May 1993.

[22] J. O. Kephart and S. R. White. Measuring and modeling computer virus prevalence. In *Proceedings of the 1993 IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, CA, May 1993.

[23] G. Kesidis, A. Singh, D. Cheung, and W. W. Kwok. Feasibility of fluid-driven simulation for atm network. In *Proceedings of IEEE Globecom'96*, November 1996.

[24] C. Kiddle, R. Simmonds, C. Williamson, and B. Unger. Hybrid packet/fluid flow network simulation. In *Proceedings of the 17th Workshop on Parallel and Distributed Simulation*, June 2003.

[25] D. M. Kienzie and M. C. Elder. Recent worms: A survey and trends. In *Proceedings of the 2003 ACM Workshop on Rapid Malcode*, pages 1–10, October 2003.

[26] M. Liljenstam, J. Liu, D. Nicol, Y. Yuan, G. Yan, and C. Grier. Rinse: the real-time immersive network simulation environment for network security exercises. In *Proceedings of 2005 Workshop on Principles of Advanced and Distributed Simulation*, June 2005.

[27] M. Liljenstam and D. Nicol. Comparing passive and active worm defenses. In *Proceedings of 2004 Winter Simulation Conference*, December 2004.

[28] 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 Workshop at ACM*, Washington DC, Oct 2003.

[29] M. Liljenstam, Y. Yuan, B. J. Premore, and D. Nicol. A mixed abstraction level model of large-scale internet worm infestations. In *Proceedings of the 10th IEEE/ACM Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS)*, Fort Worth, TX, Oct 2002.

[30] B. Liu, D. R. Figueiredo, Y. Guo, J. Kurose, and D. Towsley. A study of networks simulation efficiency: Fluid simulation vs. packet-level simulation. In *Proceedings of IEEE Infocom'01*, 2001.

[31] B. Liu, Y. Guo, J. Kurose, D. Towsley, and W. Gong. Fluid simulation of large scale networks: Issues and tradeoffs. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, June 1999.

[32] J. Liu. Dartmouth ssfnet home. http://www.crhc.uiuc.edu / jasonliu/ projects/ssfnet/.

[33] J. Liu. *Improvements In Convervative Parallel Simulation Of Large-Scale Models*. PhD thesis, Dartmouth College, 2003.

[34] J. Liu, Y. Yuan, D. M. Nicol, R. S. Gray, C. C. Newport, D. Kotz, and L. F. Perrone. Simulation validation using direct execution of wireless ad-hoc routing protocols. In *Proceedings of 2004 Workshop on Parallel and Distributed Simulation (PADS'04)*, 2004.

[35] A. Medina, A. Lakhina, I. Matta, and J. Byers. Boston university representative internet topology generator. http://www.cs.bu.edu/brite/.

[36] Mirage networks. http://www.miragenetworks.com.

[37] Defense Modeling and United States Department of Defense Simulation Office. High level architecture. https://www.dmso.mil/public/transition/hla, April 2005.

[38] D. Moore, V. Paxson, S. Savage, C. Shannon, and S. Staniford. Slammer worm dissection: Inside the slammer worm. *IEEE Security & Privacy*, 1(4):33–39, 2003.

[39] Modeling of security and systems. http://www.linklings.net/MOSES/.

[40] D. M. Nicol, M. Goldsby, and M. Johnson. Fluid-based simulation of communication networks using ssf. In *Proceedings of the 1999 European Simulation Symposium*, Erlangen-Nuremberg, Germany, October 1999.

[41] D. M. Nicol and G Yan. Discrete event fluid modeling of background tcp traffic. *ACM Transactions on Modeling and Computer Simulation*, 14:1–39, July 2004.

[42] The network simualtor - ns-2. ISI, http://www.isi.edu/nsnam/ns/.

[43] K. Perumalla, A. Park, R. F. Fujimoto, and G. F. Riley. Scalable rti-based parallel simulation of networks. In *Proceedings of the Workshop on Parallel and Distributed Simulation (PADS 2003)*, June 2003.

[44] A. Poplawski and D. M. Nicol. Nops: A conservative simulation engine for ted. In *Proceedings of the 1998 Workshop on Parallel and Distributed Simulation (PADS)*, pages 180–187, June 1998.

[45] B. J. Premore, D. M. Nicol, and X. Liu. A critique of the telecommunication description language (ted). Technical Report PCS-TR96-290, Dartmouth College, Hanover, NH, 03755, November 1996.

[46] G. F. Riley, R. M. Fujimoto, and M. H. Ammar. A generic framework for parallelization of network simulations. In *Proceedings of the 7th International Symposium of Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'99)*, pages 128–135, October 1999.

[47] G. F. Riley, T. M. Jaafar, and R. M. Fujimoto. Integrated fluid and packet network simulations. In *Proceedings of the 10th IEEE Symposium on Modeling, Analysis, and ComputerSimulation (MASCOTS)*, Oct. 2002.

[48] C. Shannon and D. Moore. The spread of the witty worm. CAIDA Analysis. http://www.caida.org /analysis/security/witty/, 2004.

[49] R. Simmonds and B. W. Unger. Towards scalable network emulation. *Computer Communications*, 26(3):264–277, June 2003.

[50] S. Singh, C. Estan, G. Varghese, and S. Savage. The earlybird system for real-time detection of unknown worms. Technical report, University of California, San Diego, 2003.

[51] Scalable simulation framework. http://www.ssfnet.org.

[52] S. Staniford. Containment of scanning worms in enterprise networks. *Journal of Computer Security*, 2003.

[53] S. Staniford, V. Paxson, and N. Weaver. How to own the internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, August 2002.

[54] W32.gnuman.worm. Symantec Security Response, http://securityresponse.symantec.com /avcenter /venc/data/w32.gnuman.worm.html, February 2001.

[55] Telescope analysis. CAIDA, http://www.caida.org/analysis/security/telescope.

[56] The spread of the code-red worm (crv2). CAIDA, http://www.caida.org/analysis/security/code-red/coderedv2_fanalysis.xml, 2001.

[57] J. Twycross and M. M. Williamson. Implementing and testing a virus throttle. In *Proceedings of the 12th USENIX Security Symposium, USENIX*, August 2003.

[58] Y. Wang and C. Wang. Modeling the effects of timing parameters on virus propagation. In *Proceedings of the 2003 ACM Workshop on Rapid Malcode*, pages 61–66, Washington, DC, October 2003.

[59] N. Weaver, I. Hamadeh, G. Kesidis, and V. Paxson. Preliminary results using scale-down to explore worm dynamics. In *Proceedings of 2004 ACM Worm Workshop*, October 2004.

[60] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. Large scale malicious code: A research agenda.

[61] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. A taxonomy of computer worms. In *Proceedings of the 2003 ACM Workshop on Rapid Malcode*, pages 11–18, Washington, DC, Oct 2003.

[62] N. Weaver, S. Staniford, and V. Paxson. Very fast containment of scanning worms. Technical report, Berkeley, 2004.

[63] M. M. Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. Information Infrastructure Laboratory, HP Lab. Bristol, June 2002.

[64] J. Wu, S. Vangala, and L. Gao. An effective architecture and algorithm for detecting worms with various scan techniques. In *Network and Distributed System Security Symposium*, 2004.

[65] V. Yegneswaran, P. Barford, and J. Ullrich. Internet intrusions: Global characteristics and prevalence. In *Proceedings of SIGMETRICS*, June 2003.

[66] J. Zhou, Z. Ji, M. Takai, and R. Bagrodia. Maya: Integrating hybrid network modeling to the physical world. In *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, volume 14, pages 149–169, April 2004.

[67] C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and early warning for internet worms. Technical report, University of Massachusetts at Amherst, 2003.

[68] C. Zou, W. Gong, and D. Towsley. Worm propagation modeling and analysis under dynamic quarantine defense. In *Worm Workshop at ACM*, 2003.

[69] C. Zou, D. Towsley, W. Gong, and S. Cai. Routing worm: A fast selective attack worm based on ip address information. Technical report, Univ. Massachusetts, Amherst, 2003.