Using Plant Epidemiological Methods To Track Computer Network Worms

Rishikesh Pande

Thesis submitted to the faculty of Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science

Dr. James Arthur
Mr. Randy Marchany
Dr. T. M. Murali

May 6, 2004
Blacksburg, Virginia

Keywords: network worms, computer viruses, plant epidemiology, spatial spread

Using Plant Epidemiological Methods To Track Computer Network Worms

Rishikesh Pande

Advisors: Mr. Randy Marchany

Dr. James Arthur

Abstract:

Network worms that scan random computers have caused billions of dollars in damage to enterprises across the Internet. Earlier research has concentrated on using epidemiological models to predict the number of computers a worm will infect and how long it takes to do so. In this research, one possible approach is outlined for predicting the spatial flow of a worm within the local area network (LAN).

The approach in this research is based on the application of mathematical models and variables inherent in plant epidemiology. In particular, spatial autocorrelation has been identified as a candidate variable that helps predict the spread of a worm over a LAN. This research describes the application of spatial autocorrelation to the geography and topology of the LAN and describes the methods used to determine spatial autocorrelation. Also discussed is the data collection process and methods used to extract pertinent information. Data collection and analyses are applied to the spread of three historical network worms on the Virginia Tech campus and the results are described.

Spatial autocorrelation exists in the spread of network worms across the Virginia Tech campus when the geographic aspect is considered. If a new network worm were to start spreading across Virginia Tech's campus, spatial autocorrelation would facilitate tracking the geographical locations of the spread. In addition if an infection with a known value of spatial autocorrelation is detected, the characteristics of the worm can be identified without a complete analysis.

Acknowledgment

I thank my advisors Randy Marchany and Dr. James Arthur for all their help and advice. My research and this thesis would never have been complete without their guidance, patience and direction. I also thank Dr. T. M. Murali for serving on my committee, as well as providing insight on various aspects of the research.

I especially thank Randy Marchany for being a friend and mentor. His belief in this research and his dedication helped me throughout my study. I learned several things from him that I will carry with me for the rest of my life.

I also express my sincerest thank you to Dr. Arthur for his conviction in my work. His insights into several topics have shaped the progress of this research.

In addition, this research would never have been completed without the diligent work of the people at Dshield, especially Mr. Johannes Ullrich and Mr. Alan Paller. I would like to extend my gratitude to them. In addition, I thank Mr. Babacar Wagne for his numerous contributions to the local Dshield project.

Last but not least, I would like to thank my parents and friends for their support and encouragement. Without them, this research would have simply remained a thought at the back of my mind.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

In the 8th annual Computer Crime and Security Survey [Computer Security Institute, 2003], 92% of all respondents reported cyber crimes and other information security breaches. Recent virus/worm attacks on the Internet have incapacitated several utility and other important services. Computer viruses caused a loss of $55 billion in 2003 [Reuters, 2004], almost double the loss in damages in previous years. The cost is expected to increase in coming years.

Early viruses spread from computer to computer via floppy disks and other manual disk installations. For example, the (c) Brain virus overwrote a part of the boot disk and took almost a year [Paquette, 2000] to spread across the world.

The Internet allows viruses and worms to spread to a large number of systems quickly. The Morris worm in 1988 was the first Internet worm and used vulnerabilities in several programs to spread. The worm infected 10% of the computers connected to the Internet at that time [Orman, 2003]. The SQL Slammer worm of 2003 [Travis *et al.*, 2003] also used vulnerability inherent in software and affected 15% of the computers connected to the Internet. However, the total number of computers connected to the Internet in 2003 was significantly greater than the number of computers connected to the Internet in 1988; thus, the SQL Slammer worm caused far greater damage.

The damage from network worms is not limited to financial losses and the public embarrassment to commercial enterprises and universities. Essential services, including utility services [Schneier,

2003] and aircraft safety [Airwise News, 2003], have been interrupted because of the effects of network worms.

Unfortunately, system and network administrators are not able to control losses because no warning signals are generated of an impending attack. Worms like the SQL Slammer worm [Travis *et al*., 2003] of 2003 spread across the Internet in 10 minutes. Most administrators did not have sufficient time to secure their networks against this worm.

Preventative measures provide the most effective protection—in this case, patching the vulnerable systems before the worm is released. Since no warnings occur, these measures are inherently reactive in nature. Further, in large companies and organizations with mobile users, applying and supporting a patch to systems can become a systems management nightmare [Bautts, 2003].

The spread of a worm depends on the characteristics of the worm (such as the exploit it uses and the IP number generation technique), as well as the susceptibility of the computer to the exploit the worm uses. Most modern day worms use a self-propagation mechanism to move from computer to computer. These worms can either self-trigger or make use of an external trigger. For example, in the case of the SoBig virus, a user had to open an infected e-mail attachment before the virus could infect the computer.

In the case of a self-propagating worm, no user action is necessary to infect the computer. Thus, the spread of these worms is rapid. Such a worm generally uses a random IP generation technique to move to its next computer victim. This type of worm uses the first one or two bytes of the IP number of the infected machine to scan for the next machine to infect. For example, the Code Red II worm had a

three-in-eight chance of using the first two bytes of the infected machine's IP address to scan for the next IP address to infect [Bautts, 2003]. In effect, IP address generation is not truly random. Determining the flow of computer worms in the network becomes critical to the counter measures selected. In some way this thinking is an oxymoron; while some computer viruses spread using random IP generation techniques, this research is trying to determine if the spread of the virus follows a pattern.

Some form of automated computer program performs most computer attacks. Knowing the network path pattern the worms follow would help build an early warning system for the subnet.

For the purposes of this thesis, I concentrate my efforts on self-propagating worms that spread from computer to computer by generating random IP numbers. The goal of this research is to find methods for predicting the track that widespread computer viruses/worms in the network will follow, so that suitable counter measures can be taken.

1.2 Problem

Every network worm exploits a different vulnerability and attempts to spread in a new way so that it can defeat the computer and network defenses while affecting the maximum number of computers. Some of these system vulnerabilities may have been announced from 1 year to 1 day in advance of the release of the worm. However, the time from disclosure of a vulnerability to the release of a virus that exploits the vulnerability is rapidly decreasing.

Moreover, each virus uses different propagation mechanisms, making it difficult to predict the kind of propagation mechanism the virus will use. A certain class of viruses will spread differently from another class simply because the two classes use different IP address generation techniques.

System administrators, though, have no warning of an impending worm until it hits their network, and often times it is then too late to react. In the case of fast propagating worms, system administrators do not get sufficient notice about the activity of a worm or a virus in the networks until after the incident has passed. Additionally, most small-scale enterprises do not have the resources to invest in round-the-clock support of their systems. Thus, if a virus spreads during off-office hours or when the administrators are not in contact range, the state of the system is volatile.

The extent of damage from each worm increases over time. Some measures to quell the damage have been in existence for some time. Most popular among these measures is an Intrusion Detection System (IDS), which is fitted with a certain number of rules to detect an

intrusion. The IDS notifies users of an intrusion, depending on their pre-defined rules, and depends on user action to stop the intrusion.

An Intrusion Prevention System (IPS) extends the IDS by coupling an IDS with a firewall or some type of limiting device to block the propagation of the intrusion. However, both IDS and IPS are plagued by a high number of false positives (detecting an intrusion when one is not present) and false negatives (failure to report an intrusion when one is present). Some IDS also have to be trained in a secluded environment to enable the IDS to recognize the normal state of the system and, thus, detect the presence of an anomaly in a real environment.

The IDS tend to perform better with larger quantities of training data than with the small amounts of data that one subnet provides. Therefore, several Internet–wide Distributed Intrusion Detection Systems (DIDS) now exist [SANS Institute, 2001; Dshield, 2001; Baldwin, 2001] and have proven to be early indicators for Internet-wide intrusion attacks.

The DIDS are widely accepted as standards for detecting intrusions on a worldwide scale. They receive data from various sources, such as personal firewall logs, enterprise IDS logs, and educational institutes. An analysis is carried out on the data and the required authorities are contacted via e-mail. As is obvious, the more widespread the data collection agents are, the better the IDS will be able to predict intrusions.

If a DIDS has a large number of widespread data collection agents, one can follow the flow of the virus. However, unfortunately, it is still not possible for the DIDS (I refer to the global Internet-wide DIDS as DIDS in this thesis) to predict the spread of self-propagating

viruses across the Internet. If this capability existed, a system or network administrator could be warned before the virus spread out of control. The administrator could change the network configurations to disallow harmful traffic from flowing into the network and, thus, save the organization from damages.

No methods are available to predict how the worm will travel. Some network and human characteristics may cause a certain ordering in the method that the computer viruses flow across the network. To complicate matters, several ways exist for looking at the network structure of a local area network. An obvious division is the hierarchical network structure of the LAN (local area network). An alternative is the geographical division, reflecting how the networks are geographically located.

If the spread of the worm can be determined by using a model, strategies to impede the spread of the virus can also be put in place, as in the case of biological epidemics. For example, during the 2003 SARS epidemic (SARS is Severe Acute Respiratory Syndrome [Center for Disease Control, 2003b]), the United States was placed on high alert after SARS cases were detected in the neighboring country of Canada. Human traffic to and from the major SARS infected areas (China, Hong Kong, Taiwan, etc.) was carefully monitored for signs of the SARS infection.

As mentioned earlier, most DIDS can detect a large-scale attack across the Internet and can immediately warn their clients. If multiple network administrators each control their individual subnets, the time required to dissiminate information to the people making the changes is vital. An aggressive notification method that is tempered by the degree of the intrusion level can resolve this issue. The previously

mentioned issue of determining the spread of the network worm needs to be resolved before the notification strategy is determined.

The problem that this research intended to solve can be summarized as:

"Determine the spatial spread of computer network worms by applying parallel strategies from plant epidemiology."

1.3 Proposed Solution

Network worms spread aggressively across the Internet, and each new worm increases the damage caused to enterprises and undermined the global economy. Network administrators are at a loss to determine when, and which, systems will be affected by these worms. Overly general policies are harmful because they block legitimate transactions from taking place. For example, a policy to stop File Transfer Protocol (FTP) applications may lead to breakdown of applications that rely on this service.

This thesis proposes one approach to addressing the problem by determining the spatial flow of the virus in LANs. Parameters and models from other fields, such as climatology and epidemiology, were examined to determine those that best match the characteristics of computer epidemics.

After the parameters were determined, firewall log data was collected from the D-shield [Ullrich, 2000] DIDS for the periods of infection for the Linux Slapper [Symantec Corporation, 2002], SQL Slammer [Travis *et al*., 2003] and MS Blaster worms [Symantec Corporation, 2003]. The Virginia Tech LAN was the focus of the analysis of the spread of the virus.

Data analysis was conducted to determine the parameters for the spatial flow of each virus. The primary parameter selected for study is the spatial autocorrelation parameter, which is widely used in plant epidemiology to predict the spread of plant viruses [Cliff and Ord, 1981]. Spatial autocorrelation was evaluated to analyze its applicability to computer networks. This parameter will assist in predicting the topological regions on the local area network that will be most susceptible to the worm.

This model was then used as part of a proposed design for an IDS. The design of such a system, currently code-named ‘Middle-range Intrusion Detection System' (MIDS), is proposed in this thesis.

Background information is given on computer and biological epidemics, how they compare to each other, the specific parameters of interest, and the reasons for these choices. Presented next are the data collection, the analysis conducted, and the results obtained. Finally, a design is proposed for an IDS, which works in collaboration with a global DIDS. Such an IDS will assist in predicting the flow of the virus and then it will take appropriate actions based on the severity level of the worm. Conclusions and future work are also detailed.

2 Background

This section introduces background information vital to this research. A definition is given for computer epidemics and causes such epidemics are listed. Section 2.2 cites a particular case of computer viruses, the network worms. Section 2.3 details the spreading mechanisms used by network worms and its effects. Network worms spread in a manner that leaves behind a signature. Section 2.4 details the cause of these signatures. Containment strategies for network worms are outlined in Section 2.5. An IDS helps determine the occurrences of network worms, and a brief overview into the major types of IDS is presented in Section 2.6. A broad overview of biological epidemiology is presented in Section 2.7. Finally, computer epidemic models based on biological epidemics are presented in Section 2.8

2.1 Computer Epidemics

The word "epidemic" has been defined as "an outbreak of a contagious disease that spreads rapidly and widely" [*American Heritage Dictionary*, 2000] In a similar manner, a computer epidemic could be defined as "a computer virus [Symantec, 2000a] or worm [Symantec, 2000b] spreading rapidly and extensively by infection and affecting many individual computers in an area or a population at the same time."

Computer epidemics were first investigated by Kephart, Chess and White [Kephart, Chess and White, 1993], who described how computer viruses propagate. It was found that computer epidemics spread in a manner similar to biological epidemics. The viruses analyzed at the time were primarily spread via floppy disks. Most viruses now spread via the Internet. Therefore, the spread patterns of current epidemics are different from the viruses that were analyzed by these investigators.

Drawing a parallel to biological epidemics, computer epidemics spread differently because viruses belong to different classes. This research primarily considers computer epidemics caused by a certain class of computer viruses called network worms. Network worms tend to spread from one host to another by producing a new random IP (Internet Protocol) address, and then attempting to infect the host that has the generated IP address. This propagation mechanism continues until all susceptible machines have been infected.

2.2 Network Worms

Robert Morris, a Cornell University computer science graduate student, released the first public Internet worm in 1988. The Morris Worm [Orman, 2003], as it became known, exploited known flaws in the finger and sendmail services of Unix systems, as well as in the common webs of trust inherent in the rlogin facility. The worm's only activity was to replicate itself in as many hosts as possible. The Morris worm infected roughly 10% of the Internet computers and cost an estimated $100 million to clean up.

Since then, several network worms have been released on the Internet with varying levels of success. Current estimates set the cost of damages due to recent worms in the billions of dollars [Reuters, 2004]. The common factor is that all are network worms. The most important characteristic that distinguishes network worms from other replicating programs is that they use the Internet backbone to spread from one computer to the next.

Different worms in the past have used various attack vectors. Some of the common attack vectors include the Remote Procedure Call (RPC) service, Instant messenger, and e-mail services [Paquette, 2000]. These services frequently are run on client machines. Since e-mail and instant messenger viruses depend on human behavior (opening of an attachment, clicking on a malicious link), the spread of e-mail viruses are highly dependent on the contact structure of human beings. On the other hand, self-propagated viruses depend on the existence of a flaw in the computer itself—no human action is necessary.

2.3 Spreading Mechanisms

The spreading mechanism is responsible for the selection of the next target of infection for the worm. Several different methods exist for the spread of viruses [Balthrop *et al.*, 2004], however, network worms transmit by generating random IP addresses. Worms use various techniques to generate a random IP address. A good pseudo-random number generator with a good starting seed is a typical method used to produce an IP address of the form A.B.C.D where A, B, C, and D are random numbers between 0 and 255. A variant of the Unix srand () function [Unix manual pages, 2003] is a popular method for producing the initial seed. However, this method has been shown to have deficiencies in the way that seed is generated [Garfinkel, Spafford and Schwartz, 2003]. As a result, the random generation technique does not produce IP addresses over a wide range. Thus, virus writers take care while writing the generator because, if the seed is not random, the IP addresses generated may be only a partial set and, thus, adversely affect the spread.

Recent viruses have used new methods of generating a good random seed and have used more conservative approaches, such as restricting the randomization to the same sub-network as the infected computer. This method has also proven to be quite successful, as in the case of the Blaster worm. The worm writer used a random number generator that produced an IP address within the same Class B subnet 40% of the time and a completely random IP address 60% of the time.

However, most IP addresses that are formed randomly by the above method do not have live hosts connected to them. In fact, most system administrators utilize only 30% of the allocated address space [Bautts, 2003]. Therefore, a large number of infection attempts do not

produce a result. Consequently, the only way a virus can cover the entire community of the Internet is by producing a large number of infections as fast as possible. This strategy leads to instability in the very medium that the virus uses for transmition: the Internet. Previous worm occurrences have shown that a Blaster-style worm mostly affects the working of the infrastructure on the Internet.

The SQL Slammer worm infected 15% of all hosts connected to the Internet. The amount of traffic generated by this worm caused many routers to fail and crash. Ironically, it stopped its own growth by overwhelming the network. However, the SQL Slammer was deemed the fastest spreading worm in history because of the connectionless protocol it used to transmit the infection (UDP) and its relatively small size (369 bytes).

All other worms released prior to January 2004 used the connection oriented, TCP (Transmission Control Protocol) protocol, which requires the targeted computer to respond to the infection computer. Once the target computer produces a response, the infected computer transmits the malicious code, which tries to exploit vulnerabilities in the computer software. If no computer exists at the generated IP or the port is closed, the virus receives no response.

In order to reach as many computers as possible, a network worm uses a multi-threading approach in attempting to infect a pre-defined finite number of computers. The code for the Blaster worm produced a maximum of 20 threads that attempted to infect other computers. If the computer at the requested IP responded, an infection was attempted. If no message was received after 5 minutes, the thread terminated.

Even if no computer exists at the random IP chosen, a TCP SYN packet is still transmitted on the network. This large number of packets on the Internet causes a SYN flood, which is a form of denial of service attack because network devices receive information faster than they can process it. Due to this heavy flood, normal packets cannot be sent on the Internet; thus, valid information fails to be delivered.

This heavy flood of network activity can also cause critical systems to collapse and cause damage to emergency and utility services [Schneier, 2003]. Such a collapse in primary infrastructures of the Internet adds to the problems occurring in the spread mechanism. In addition to heavy traffic, some of the available routes shut down. This paucity of available routes leads to an avalanche effect , and the entire Internet comes to a standstill.

The Warhol worm [Staniford, Paxson and Weaver, 2002] can spread across the Internet in 15 minutes to one hour. The spreading mechanism of such a worm is highly specialized because it uses hit lists and predefined IP address ranges. However, no practical implementation of a Warhol worm has occurred although some experts [Broersma, 2003] contend that the SQL Slammer worm was an implementation of such a worm.

2.4 Signatures

Network worms exploit vulnerabilities in the software on the resident system to connect to other machines with the purpose of infecting them. As worms propagate from one machine to the next across the Internet, they use the same method to move from the first computer to the second, from the second to the third, and so on and so forth. This method invariably leaves a pattern on the network where the infecting computer is located. Such a pattern is called a network signature. A network signature is a pattern in Internet traffic that identifies the traffic flowing through the network. When this signature is referred to as an intrusion attempt, such as that made by a worm, it is called an attack signature. Attack signatures represent the specific bits of program code that each virus or worm carries [Salkever, 2001].

The following example illustrates this point further. If traffic on the street is observed from the top of a skyscraper, the type of car on the street can be identified– coupe, mini-van, truck, etc., but it would be very difficult to identify the exact car (license plate, occupants, etc). However, if the observation details a line of 3 coupes, followed by a mini-van, followed by 2 trucks over and over again, such a pattern is easily recognizable. The pattern of 3 coupes, a mini-van, and 2 trucks would form a signature.

In the same way, a network administrator cannot identify every single packet flowing through the core routers. However, if the administrator sees strings of suspicious packets forming a signature over and over again, the cause of the signature should be inspected. If the packets within the trace are an attempt to attack another computer, this signature would form an attack signature.

Network signatures can be primarily detected in two separate ways [Frederick, 2002]: packet grepping and protocol analysis.

Packet grepping is a static intrusion detection method; the IDS performs pattern matching on individual packets flowing through the network with predefined signatures. This method is static because the state of the connection is not stored. However, new or unknown intrusions will not be tracked in this method.

Protocol analysis is a slightly more advanced method of intrusion detection; an initial or stable state of the system is defined. Instead of checking individual packets, the IDS checks entire connections for determining what the source system is attempting based on predefined signatures. If the attempt is something the IDS has seen before and deems as a normal connection, it allows the connection to flow through. If not, the IDS raises an alert. As can be inferred, an approach performing stateful packet inspection is slower and more complicated; therefore, it is sparingly used.

The attack signatures for a worm may be different for both approaches. Protocol analysis generally requires complicated attack signatures called profile-based signatures, based on Markov models. Packet grepping requires attack signatures with transport protocol (TCP/ UDP) information, that is, important characteristics such as port number, packet length, window size, etc. Packet grepping, however can block valid traffic if that traffic has characteristics matching the attack signature.

Attack signatures are viewed differently, depending on the IDS and the developer of the IDS. Most IDS have the facility to construct a user-defined attack signature if some unwanted traffic is noticed on the network. For example, a particular network wanting to stop all telnet

requests coming in after a port scan can configure an attack signature based on these characteristics in the IDS and distributes it. A sample network signature from the Snort [Snort Network Intrusion Detection System, 2000] network signature database is shown in Figure 1. This sample network signature shows different TCP flags that the IDS checks; if matches occur, the log message on the last line is recorded and displayed in the event log.

```
Name:           Blaster Worm Command Shell Attack V1 -NG
Group:          Host Services
Active-Flag:    1
Priority:       3
Input-Source:   1
Dest-Port:      4444
Scan-String:    tftp *0* GET msblast.exe
Scan-End:       0
Case-Sensitive: 1
Compact-Spaces: 0
Action:         4
Log-Message:    [~SENSOR] - Blaster Worm Command Shell Attack from ~SRCIP
```

Figure 1. Blaster command shell attack signature

2.5 Containment

Most computer viruses including network worms initially spread slowly, achieve a critical mass, and then spread rapidly across the Internet. The number of new infections eventually levels for one of two primary causes: the worm achieves its critical mass or the preventive measures become effective. A worm reaches its critical mass when it has infected as many computers as it can by using a random scanning algorithm. Network administrators generally initiate preventive measures when they are informed of a worm. Figure 2 illustrates the number of infected hosts in a subnet of the Internet during the Code Red v2 epidemic of July 2001 [Moore and Shannon, 2001].

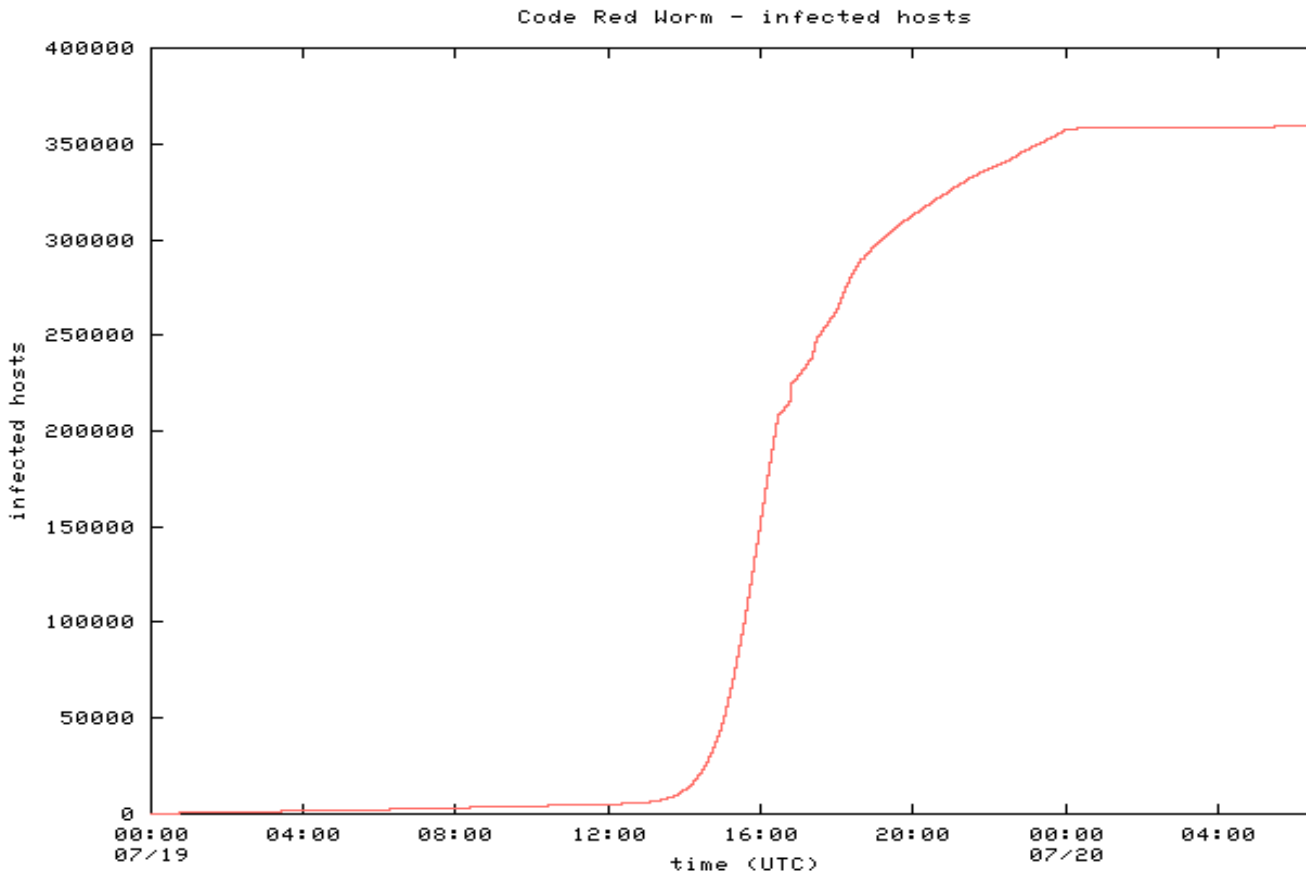


Figure 2. Code Red v2 time-series spread

Korzyk [1988] developed a time-series forecasting model for predicting Internet security threats. This model is based on the assumption that after a certain time interval, the Internet weathers the effects of a network worm. Further work in this area has led to an effective measurement methodology for a time-series model of a network worm [Yurcik, Korzyk, and Loomis, 2000].

The goal of worm containment, in a broad sense, is to recognize (either on end systems or in the network traffic) that a worm is spreading and to disrupt its spread before it can cause widespread harm [Staniford, 2003]. However, this goal cannot always be achieved, and counter measures, including patching, upgrading, firewalling and disconnection after a worm incident, are still widely used.

In principal, counter measures can quickly reduce, or even stop the spread of infection, thereby mitigating the overall threat and providing additional time for more heavyweight treatment measures to be developed and deployed.

Cleaning up after a worm incident is an example of reactive security in an organization. However, proactive security, including immediate upgrading and patching of systems has proven costly in some enterprises because it may break some critical running applications. A mix of proactive and reactive security is considered a good approach where new patches and upgrades are verified on test systems before being applied to the critical system [Cheswick *et al.*, 2003].

Based on the classical Kermack-McKendrick epidemic model [Freunthal, 1980], researchers have formulated a two-factor model [Zou *et al.*, 2002] for the spread of a network worm. This model leads to a better understanding and prediction of the scale and speed of the

spread of network worms. This model takes into account the human countermeasures including removing both susceptible and infectious computers from the Internet after systems and network administrators learned of the spread of the Code Red v2 worm. As more people learn of a worm, an increasing number implement some form of counter measure–patching or upgrading susceptible computers, setting up firewalls on edge routers, or simply disconnecting their systems from the Internet.

Two primary containment strategies have proved effective [Moore *et al.*, 2003]: content filtering and address blacklisting. Address blacklisting is an extension of spam lists where specific IP addresses are blacklisted and all transactions from that specific IP are dropped. Content filtering is an advanced technique where the network packets, which contain malicious code, are identified and dropped. Intrusion Detection Systems generally use some form of a content filtering mechanism. Bayesian filtering algorithms have been used for content filtering and generally yield a better output than address blacklisting methods by an order of magnitude. However, they are also computationally intensive.

Nojiri *et al*. [2003] propose a friend system for co-operatively mitigating huge intrusion incidents such as a worm epidemic. Cooperating members communicate with others by a "friend protocol" that spreads attack reports to potentially vulnerable uninfected sites. However, during worm epidemic conditions, the network itself is under duress. Adding more traffic to the networks creates a race condition where protocol traffic and worm traffic compete for the same resources.

Williamson [2002] suggests using virus throttling for worm containment. Virus throttling involves observing the type pf connections

that a machine makes with a new machine under normal conditions (generally low level below 0.5 – 1Hz). In the same manner, a worm makes high-level connections to new machines (for example, the Code Red virus made connections at the rate of approximately 200Hz [Williamson, 2002]) Once these facts are gathered, additional software can be implemented in the network stack of an operating system that limits the rate that a host can make outbound connections to new hosts (new in the sense of not being in a cache of recently visited hosts). An obvious limitation of this technique is that, if the worm is able to get system access on the host, it can disable the filtering mechanism and, thus, spread unchecked.

The Counter Malice product [Silicon Defense Inc., 2002] operates by extending the research of Williamson [2002] to a network device. The device makes the network more resilient to the worm; however, the worm must first infect the network that has the device before the device can impede the flow of the virus.

2.6 Intrusion Detection Systems

Several methods to counter the emergence of worms have been investigated. Prominent among these are the network-based IDS, which monitor the network for any suspicious activity. When such an activity is discovered, it is immediately reported via a pre-determined notification method.

The idea of a DIDS has been considered since the late 1980s. However, not until the Internet connected computers across the globe was the importance of correlated data from various agents understood. Correlated data helps in understanding major occurrences of intrusions. For this reason, large-scale DIDS were developed in the late 1990s [Inella, 2001]. Robbins [2002] outlines the primary reasons for moving from individual IDS to a DIDS.

Distributed Intrusion Detection Systems proved effective in the rapid assessment of virus activity across the Internet. A good example is the detection of the 2001 Lion worm at the Internet Storm Center at the SANS (SysAdmin, Audit, Network Security) Institute [SANS, 2001a]. Distributed Intrusion Detection Systems are now widely accepted as standards for detecting intrusions on a worldwide scale. They receive data from various sources, such as personal firewall logs, enterprise IDS logs and educational institutions. The data are analyzed and the required authorities are contacted via e-mail. This strategy helps many ISP and domain owners to identify computers on their networks with malicious software. An example of such a system is shown in Figure 3.

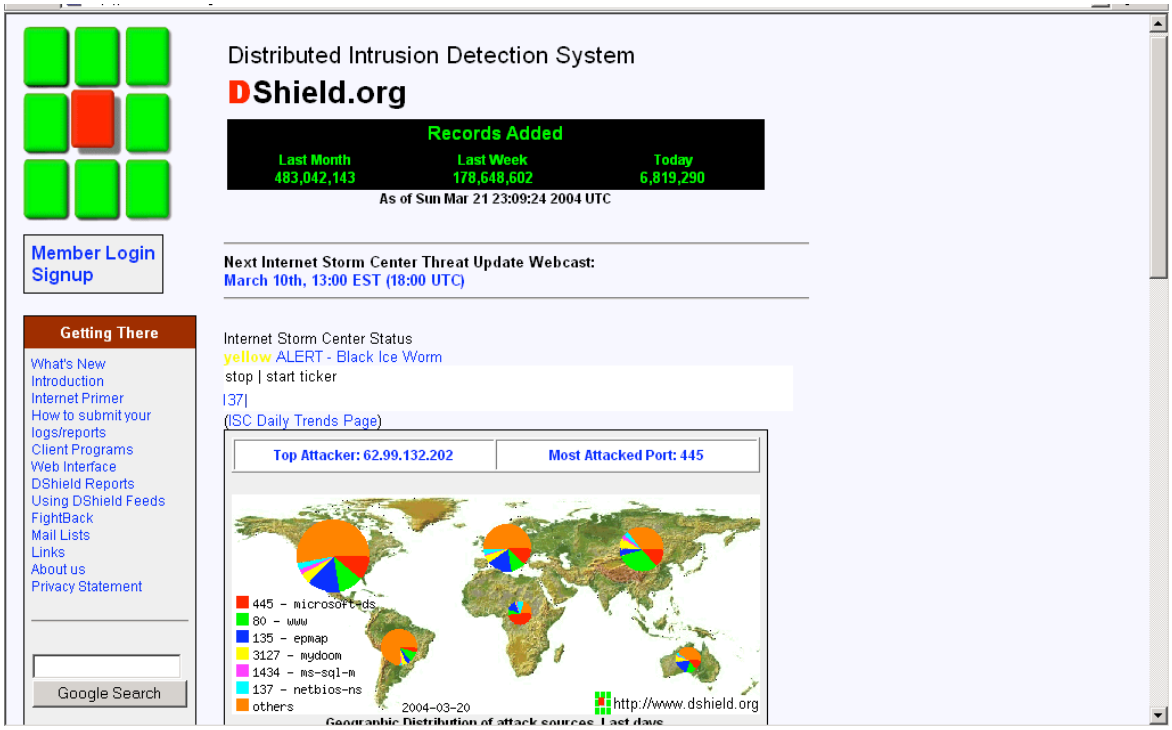


Figure 3. Dshield Distributed Intrusion Detection System

Unfortunately, DIDS still cannot predict the spread of viruses across the Internet. However, the detection and prediction of the movements of these viruses within a LAN the data management systems need in deciding their next course of action. For example, if subnets P and Q have been attacked and a prediction mechanism indicates the possibility of subnets A and B being under immediate threat, the management system can choose to alert the network administrators of those subnets with an emergency message. At the same time, the network administrators of the already infected subnets can be notified to start disaster recovery.

The problem facing many LAN is the rapid dissemination and transfer of information concerning attacks. Most DIDS can detect a large-scale attack across the Internet and warn their clients immediately. However, if multiple network administrators each control

their individual subnets, the time required to disperse information to the individuals who can make the changes is vital.

2.7 Biological Epidemiology

The Center for Disease Control (CDC) defines epidemiology as "the study of the distribution and determinants of health-related states in specified populations, and the application of this study to control health problems" [CDC, 2003a]. Depending on the population it affects, epidemiology can span various fields, such as plant epidemiology, human epidemiology, and animal epidemiology. Because the general analytical methods and techniques applied to epidemiology can be the same across different populations, some methods can span all fields. Consequently, intense investigations in the field of epidemiology have attempted to determine the causes of an epidemic and the solution to its problems. However, since epidemics affecting each species (e.g. plants, animals, and humans) have different characteristics, different analytical methods are preferred for each. This present research has led to a significant understanding of epidemiological models.

Epidemiological models are primarily classified into three categories: box or compartmental models, continuous space models, and network models. Box models divide populations into compartments, and the spread of disease across the compartments is determined if possible. These models assume that contacts within a compartment are homogeneously distributed. Box models have been the standard for dealing with spatial heterogeneity in human populations [Anderson and May, 1984]. Many of the results from box models are applicable to human and animal populations.

Continuous space models represent the surface of Earth as a plane and therefore simplify the mathematical description of space [Bolker, 1997]. They assist in qualitatively understanding the processes governing spatial spread. Lattice models have been used with powerful

analytical tools to approximate the dynamics of the epidemic in a more realistic spatial setting. The grid cell structure of lattice nodes has been successful in using fixed dispersion methods for predicting the spatial movements of plant epidemics. These dispersion methods depend on infection vectors such as wind, animals, and spores.

Network models combine the most realistic components of the continuous-space and the box models. These models have helped construct approximations of the true contact structure of humans [Bolker, 1997]. Network models have been very useful in determining the spatial spread of human epidemics. However, they are also the most computationally and analytically difficult models to construct.

2.8 Computer Epidemic Models

Kephart *et al.* [1993] used lattice models with directed graphs to track the spatial spread of early computer viruses. Each computer is considered as a node in the model. The computer where the infection started was considered in the center. All connections to each node were the computers that modeled the human relationship between the owners of he computers. This type of model was applied to a lattice structure with three neighbors. This lattice structure resembled a cube, which formed a grid cell of eight. It is observed that there is clustering in the center. This formation indicated that the spread is limited to a circular region around the first incidence of the virus. However, the viruses under consideration by Kephart *et al.* [1993] were boot sector viruses, which spread via human contact structure. The above model is no longer valid because current viruses use random scanning of IP addresses to determine the next target. However, the Kephart *et al.* [1993] research serves as the starting point and the main inspiration for the present research.

Related work in the field of computer virus prediction has concentrated on game theory for predicting the actions of attackers on individual computers and for predicting when an attack might occur [Liu and Li, 2001]. Recent research has concentrated on fuzzy logic [Botha and Solms, 2003] and behavioral assessment to predict attacks. However, most of these techniques are only practical for determining attacks on individual computers.

Based on the simple epidemic model [Daley and Gani, 1999] and the Kermack-McKendrick threshold model [Kermack and McKendrick, 1932], a two-factor worm model was presented by Zou *et al.*. [2002]. The model accurately follows the spread of the 2001 Code Red worm. The two-factor model helps predict the temporal

characteristics of the worm and shows an exponential rise in the spread of the worm in the early stages when the networks have relatively normal levels of traffic flow. At this stage, most computers are not yet infected with the worm. Zou *et al.* [2003] suggested using a Kalman filter [Kriedl and Frazier, 2003] on the LAN to detect the propagation of the worm at an early stage. The filter is based on observed illegitimate scan traffic. The Kalman filter is an exponential filter, which increases quarantine strategies as the amount of traffic on the Internet increases. However, this filter requires the transmission of status from agents across the network at a time when the network is heavily loaded with worm packets. If the agents are not able to communicate, the filter fails.

This chapter has presented the background necessary for understanding the core concepts discussed later in this thesis. Discussion continues on the similarities and differences between plant and computer epidemiology and the choice of the particular statistical variables and models used in this analysis.

3 Computer vs. Biological Epidemics

Epidemic models in earlier work on the spread of a computer virus concentrated on human epidemics. However, most network worms do not spread in many ways similar to human epidemic models, primarily due to the static (immovable) nature of computers as opposed to the flexible (movable) nature of humans. Infected humans can spread the virus spatially by their movement to a susceptible population [Anderson and May, 1985; Morris, 1994].

Another major shortcoming of earlier, worm-oriented, epidemic models [Staniford *et al.*, 2002; Zou *et al.*, 2002] is that they were not spatial in nature, i.e., these models did not support the estimation of spatial flow of a computer worm within the computer network. Plant epidemic models, however, are inherently spatial because they are used to develop preventive measures. They have a wide variety of spatial characteristics and models that can be inherently applied to computer networks.

This section outlines the primary reasons for using plant epidemiology in this analysis. Parameters for this research are also outlines, along with their importance.

3.1 Similarities

Staniford *et al.* [2002] provided a proof of concept for network worms following the Susceptible-Infected (SI) model for the spread of biological epidemics. The number of vulnerable hosts infected at time t follows the equation

$$\alpha = \frac{e^{vS(t-T)}}{1+e^{vS(t-T)}}$$

where α is the number of hosts infected at time *t*, *v* is the vulnerability density (number of susceptible computers), *S* is the scan rate of the virus, and *T* is the time when all vulnerable computers become infected. Zou *et al*. [2003] further modified this equation to adjust for the preventive measures taken by administrators after they became aware of the worm. The above equation gives rise to a characteristic S-shaped form of the infection incidence as a function of time. Figure 2 (in Section 2.5) depicts the number of computers affected by the Code Red v2 virus as observed at the Cooperative Association for Internet Data Analysis (CAIDA) center; it is an S-shaped curve.

Temporal logistic analysis of plant epidemics also produces a similar equation [Campbell and Madden, 1990].

$$y = \frac{1}{1+Be^{-r_L t}}$$

where y is the disease incidence, t is the time, r_L is the apparent infection rate, and B = $(1-y_0)/y_0$ where y_0 is the number of plants infected at the start when t = 0. This equation produces an S shaped

curve similar to that seen in the case of the Code Red worm. The time-series equations indicate that plant and computer epidemics are similar in their temporal characteristics. The time-series characteristics do not have an effect on the spatial nature of the infection. However, these similar characteristics indicate that similar spatial characteristics exist.

Wassenar and Blaser [2002] describe the similarities between plant and computer epidemics and illustrate how the evolution of virus strains in computers follows the well-known development patterns of strains in biological viruses. Their work was inspired by parallels that scientists drew between the proliferation of computer viruses and the spread of agricultural catastrophes such as the Dutch Elm disease [Lemos, 2004]. Virulence genes in plants are constantly “stolen” and reused, similar to the techniques employed by computer worm creators.

Plant epidemics reflect the way in which computer virus spread because plant epidemics also depend upon external factors, such as wind, rain, and spore dispersal, to spread the epidemic. These factors have analogous counterparts in computer virus epidemics. Continuous epidemic models are generally used to model the spread of plant epidemics. These are generally lattice models where grid cells and lattice nodes are used for modeling. Due to the basic nature of the computer network architecture, all the computers in a grid cell, the nearest neighbor, and the lattice nodes can be easily identified.

Three phases are distinct in plant epidemic spread: liberation, transport, and deposition. All three phases depend on external conditions. Wind and rain change the liberation phase, affect transport through the medium, and randomly distribute the deposition of the infection vector (Figure 4).

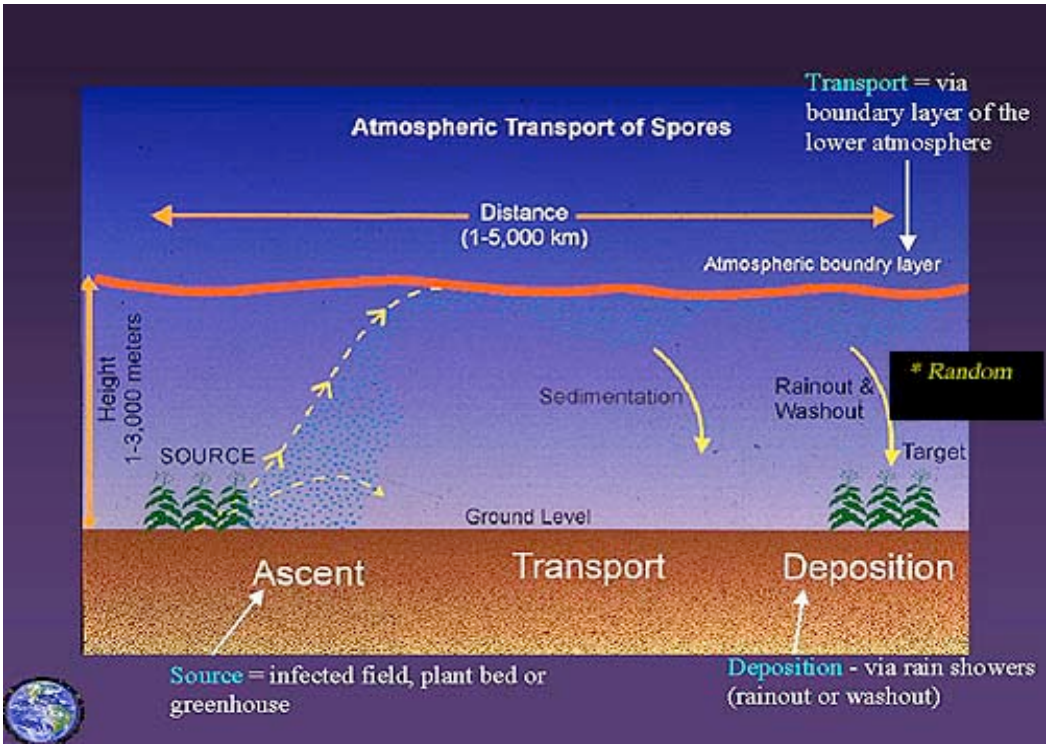


Figure 4. Virus transport mechanism in plant epidemics

Just as air is used as the transport mechanism for most epidemics, so the Internet is used as the transport mechanism for a majority of viruses in the Internet area. Wind and rain affect the transport mechanism in plant epidemics by causing changes in the transport medium, resulting in a random pattern of deposition. Subsequently, nothing ensures that the infection vector will land on a susceptible plant or even on a plant at all (the infection vector may land in a neutral, uninfectable place). In the same manner, the random IP generation mechanism in computer epidemics generates IP addresses without any certainty that the end point exists. The virus transfer mechanism in computer networks is given in Figure 5.

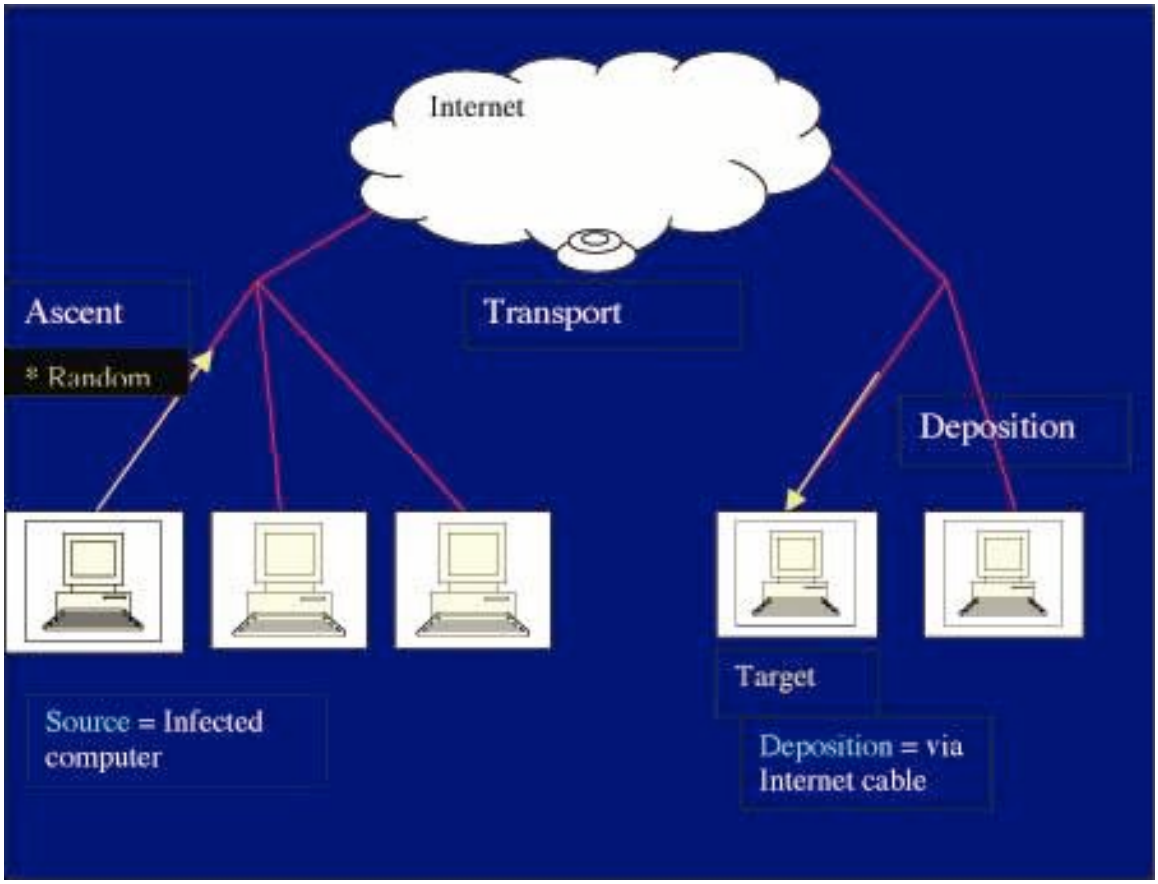


Figure 5. Worm transport in computer networks

Another point worth noting here is that both plants and computers are immobile by their very nature. Therefore, they require other mechanisms to spread their infection. The infection vector does not infect plants that are protected against the disease. Similarly, a virus cannot infect computers that have been patched against vulnerability in the software.

The stationary nature of computers also allows for easy characterization of computers into subnets similar to quadrant structures used in agricultural fields. Table 1 gives similarities between plant and computer epidemics.

Plant Epidemics	Computer Epidemics
Virulence genes "stolen" and reused	New worms reuse old software code
External environmental factors influence spread of disease	External computer network factors influence spread of network worms
Temporal characteristics of plant epidemics $y = \frac{1}{1 + Be^{-r_L t}}$ y = disease incidence, t = time, r_L = the apparent infection rate, B = $(1-y_0)/y_0$, and y_0 = number of infecteds at t=0.	Temporal characteristic of computer epidemics $\alpha = \frac{e^{\upsilon S(t-T)}}{1 + e^{\upsilon S(t-T)}}$ α = fraction of hosts infected at time t, v = vulnerability density, S = scan rate of the virus, and T = time when all vulnerable computers get infected

Table 1. Similarities between plant and computer epidemics

3.2 Differences

Although similarities can be noted between computer and biological epidemics, some fundamental differences also exist. These differences can be identified in the transmission medium and in the availability of a recipient.

The transmission medium used by the infection vector to move from one host to another in computer networks has physical limitations, which have often been reached. For example, most computer networks reached their maximum capacity during the January 2003 Slammer epidemic. On the other hand, the transmission medium for some

biological epidemics, such as air, can only become theoretically saturated. Thus, insights into how saturation of the transmission medium affects the spread of the infection vector cannot be ascertained from biological epidemiology.

In addition, the most important condition for infection in biological epidemics is that the infection vector reaches a susceptible plant. For computer epidemics, two extremely important conditions must be met: the infection vector has to reach the intended recipient and the susceptible parameter must be present on the host.

Although these differences can change the values of the parameters in epidemic models, the concept of the two models still overlap. As noted, this research only analyzed one particular class of computer viruses, network worms.

3.3 Extending Biological Parameters

The next step in this research was to choose which parameters from plant epidemics would be most helpful in determining the spatial relation between virus incidences. Since the spatial spread of the disease is the focus of this research, characteristics are modeled that help determine a spatial dependence (e.g. spatial autocorrelation and spatial clustering index). Spatial methods often included in the analysis of plant epidemics are spatial autocorrelation and the Spatial Analysis by Distance IndicEs (SADIE) clustering index.

The SADIE clustering index [Perry, 2001] determines the spatial relation among entities in point-referenced data. Aggregated, patchy, contagious, clustered, and clumped are descriptors that all refer to patterns arising from positive or attractive interactions among such individuals in point-referenced data. Thus, the index determines the presence of clusters in a field.

Spatial autocorrelation [Cliff and Ord, 1981] measures the extent to which the occurrence of an event in an areal unit constrains, or makes more probable, the occurrence of an event in a neighboring areal unit. In simple terms, spatial autocorrelation is based on the first law of geography [Tobler, 1970]: near objects are more similar than objects that are more distant.

Application of spatial autocorrelation to local area networks identifies the relation between two class C networks (a class C network is one in which the first 16 bits of the 32 bit IP address is fixed [for example, 128.173.xxx.xxx]) that receive the same intrusion or receive the same virus. Spatial autocorrelation can be quantified by various methods. This research computes spatial autocorrelation by plotting

the worldwide metrics, Moran's I values, for spatial autocorrelation [Moran, 1948].

An important consideration in spatial autocorrelation is its high dependence on the size of the area designated as a group. Unless the correct group size is chosen, spatial autocorrelation may not be evident. Since a natural topological divide exists in computer networks, each class C network can be designated as a quadrant.

Thus, two different aspects of the network, topology and geography, are used as the main underlying principles for observing autocorrelation in the incidences of network worms in a subnet. The entire class B (class B network is one in which the first 8 bits of the 32 bit IP address is fixed, for example, 128.xxx.xxx.xxx) network (analogous to a field in plant epidemiology) can be construed as a 16 x 16 grid structure with serial ordering of the class C networks. Instead of the serial ordering, neighbors are identified by their geographical proximities in the geography-based analysis. Similarly, neighbors are identified by distance with respect to network distances (hop counts) in the topology-based analysis.

Each subnet within the LAN can be denoted as one lattice node that is arranged on a lattice model with 254 possible neighbors (since a class B network can contain a maximum of 255). However, the meaning of the word "neighbor" can be further expanded to include and indicate exactly which lattice nodes are close to each other. Two re-definitions are currently under consideration: neighbors can be determined by a geographical measure or by a measure based on topological structure.

Since neighbors are close to each other, the distance between topological neighbors can be determined from the network hop count

and the geographical distance based on the physical distances between the buildings in which the subnets are located. The next subnet need not be the neighbor of a lattice node; in fact, the neighboring node may be geographically and topologically distant.

4 Evaluation Parameter

Spatial autocorrelation is the evaluation parameter used in this research. Different methods are available for calculating spatial autocorrelation. Certain factors must be considered, and some extensions must be made. Neighborhood relations are defined next and how the definitions of neighborhood are extended to fit computer networks.

4.1 Spatial Autocorrelation

Spatial autocorrelation is defined here as the relationship among values of a single incidence (in this research, the number of infections) that comes from the arrangement of the areas in which these values occur. It measures the similarity of objects within an area, i.e., the degree to which a spatial phenomenon is correlated to itself in space [Cliff and Ord, 1981]. In addition, spatial autocorrelation determines the level of interdependence between the values of the variables, and the nature and strength of the interdependence.

In the context of plant epidemiology, spatial autocorrelation tests whether the extent of infection in a particular location depends on the existence of the infection in its neighbors. In plant epidemiology, spatial autocorrelation answers one simple question: is a particular location developing a plant epidemic because its neighbors have the same infection? Extending the same logic to computer epidemiology, the question becomes: is the incidence of a network worm in a particular location dependent on its neighbors being infected with the same worm?

Spatial autocorrelation is an assessment of the correlation of a variable (in this research, extent of infection) with reference to its location, i.e., where it occurs in the space of the current consideration. Spatial autocorrelation assesses whether the values in a given space are interrelated and, if so, determines if there is a spatial pattern to the correlation, i.e. does spatial autocorrelation exist.

Spatial autocorrelation can be positive or negative. No spatial autocorrelation is exhibited when the neighbors of a grid square under consideration have different properties from those of the grid square

itself. A checker board (such as one shown in Figure 6) is a good example of the absence of spatial autocorrelation. Black squares indicate the presence of an infection while the white squares indicate the absence of infection. The squares adjacent to each white square are black, indicating that the nearest neighbors are not similar. Under these conditions, only the adjacent cells that have a common side as neighbors are considered. Such a neighborhood definition is called the rook's spatial contiguity definition.

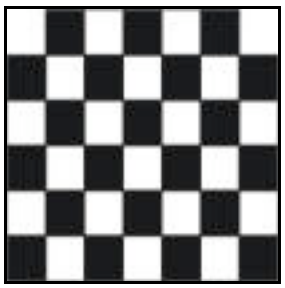


Figure 6. No spatial autocorrelation

The highest spatial autocorrelation exists when all squares have the same property. In Figure 7, the color black represents all grid squares. This example represents the maximum spatial autocorrelation possible.

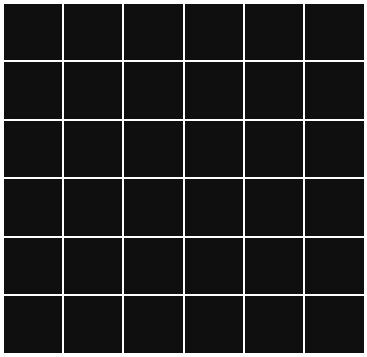


Figure 7. Highest value of spatial autocorrelation

4.2 Measuring Spatial Autocorrelation

Spatial autocorrelation can be computed by using some index of covariance for a series of distances (or distance classes) from each point. The resulting correlogram (correlation diagram) illustrates autocorrelation at each lag distance. Membership in a given distance class is defined by assigning a weight to each pair of points in the analysis; typically this weight is a simple indicator function, taking on a value of 1 if within the distance class and all else 0 (weights may also be defined in other ways, in which case they can take on real values). The weighting matrix for the geographical and the topological cases in this research is defined separately, and is dealt with in greater detail in Section 5.4.

Spatial autocorrelation exists when a systematic spatial variation occurs in the values of a given variable. This variation can exist in two forms, positive or negative spatial autocorrelation. In the positive case, the value of a variable at a given location tends to be similar to the values of that variable in nearby locations. If the value of some variable is low in a given location, the presence of spatial autocorrelation indicates that nearby values are also low. Conversely, negative spatial autocorrelation is characterized by dissimilar values in nearby locations. For example, a low incidence value may be surrounded by high values in nearby locations when negative spatial autocorrelation exists.

In this research, a high spatial autocorrelation value indicates that if an individual entity showing the existence of infection will have neighbors with similar values. The implication is that if the neighbors are not currently infected have a high likelihood of showing infection in the future. The value of spatial autocorrelation can be used to predict the likelihood of the infection spreading to neighbors of the entity itself.

The spatial pattern of a distribution is defined by the arrangement of individual entities in space and the relationships among them. The capability of evaluating spatial patterns is also a prerequisite to understanding the complicated processes underlying the distribution of a phenomenon. As such, statistics of spatial autocorrelation provide a useful indicator of these spatial patterns.

There are many indicators of spatial autocorrelation [Cliff and Ord, 1981;Chou, 1997;Goodchild, 1987;Haining 1990]:

1. Global indicators of spatial association, join count statistics; Moran's I [Moran, 1948] and Geary's c [Geary, 1954]
2. Local indicators of spatial association—LISA Gi [Anselin, 1995] and Gi* statistics [Ord and Getis, 1995]
3. The variogram approach to spatial association [Matheron, 1963] for geostatistical perspective

In this study, spatial autocorrelation calculations used the global Moran's I [Moran, 1948] methods. These statistics indicate the degree of spatial association as reflected in the data set as a whole. This research was primarily interested in an estimate of the autocorrelation over the entire space under consideration. Moran’s I computations gave values between -1 and +1; -1 is the maximum negative spatial autocorrelation detected and +1 the maximum positive spatial autocorrelation.

In plant epidemiology, Moran's I is the most commonly used coefficient in univariate autocorrelation analyses [Diniz-Filio and Telles, 2002]. Also, Moran’s I allows geographic distances to be partitioned into discrete classes. In computer networks, most computers and subnets lie in discrete subnetworks. This method better supports

analysis of spatial autocorrelation. The common use of Moran's I in plant epidemiology also influenced its choice in this research.

4.3 Calculating Spatial Autocorrelation

To obtain the spatial autocorrelation coefficient of a variable, the values of that variable must be correlated for pairs of localities. However, only those pairs of localities that are considered neighbors were correlated. These neighbors are designated as the weighting matrix. Each row of the matrix represents the neighborhood relation of the spatial place under consideration to the other grid spaces. For example, in the sample grid in Figure 8a, the corresponding row in the neighborhood matrix for grid-square 5 (considering queen's neighborhood definition) is shown in Figure 8b. Queen's neighborhood definition includes grid spaces that share a vertex or a side with the grid under consideration.

0	1	2
3	4	5
6	7	8

Figure 8a. Grid space layout

	0	1	2	3	4	5	6	7	8
5	*0*	*1*	*1*	*0*	*1*	*0*	*0*	*1*	*1*

Figure 8b. Weight matrix for grid-square 5 considering grid space in 8a

A very simple weighting scheme is used in the above example. Every neighbor who has either a side or a vertex in common with the grid square under consideration (5) is assigned a weight of 1; all others are assigned a weight of 0.

The weighting matrix is adaptable so that it reflects the situation under consideration. The weighting scheme used in the research for the two evaluation cases, geographical and topological, are explained in Section 5.4.

Moran's I is the weighted product correlation coefficient dependent upon the weights assigned which depict proximity. Moran's I coefficient is calculated as

$$I = \frac{N \sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} W_{ij} Z_i Z_j}{S_o \sum_{i=1}^{N} Z_i^2}$$

where N equals the number of regions, w_{ij} is a weight denoting the strength of the connection between areas i and j, z_i is the rate in region i centered about the mean rate (using $z_i = x_i$ – ave (x); x_i is the rate in region i); and S_0 is the sum of the weights.

S_0 is represented as

$$S_0 = \sum_{i=1}^{N} \sum_{j=1}^{N} W_{ij}, i \neq j$$

where W_{ij} is a weight denoting the strength of the connection between areas i and j.

The null hypothesis for the Moran's I calculation assumes that the area under consideration is not correlated. The expected value of I under the null hypothesis is

$$E(I) = \frac{-1}{(N-1)}$$

where N is the number of sample values being considered.

The significance (*p*) of the I value calculated is determined by an asymptotic, normal bell shaped curve. The shape of the curve is determined by the value of variance. Variance is calculated as

$$Var_R(I) = \frac{N\left[(N^2-3N+3)S_1 - NS_2 + 3S_0^2\right] - b_2\left[(N^2-N)S_1 - 2NS_2 + 6S_0^2\right]}{(N-1)^{(3)}S_0^2} - E(I)^2$$

where

$$b_2 = m_4 / m_2^2$$

$$m_4 = 1/N \sum_{i=1}^{N} z_i^4$$

$$m_2 = 1/N \sum_{i=1}^{N} z_i^2$$

$$S_1 = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N, j \neq i} (w_{ij} + w_{ji})^2$$

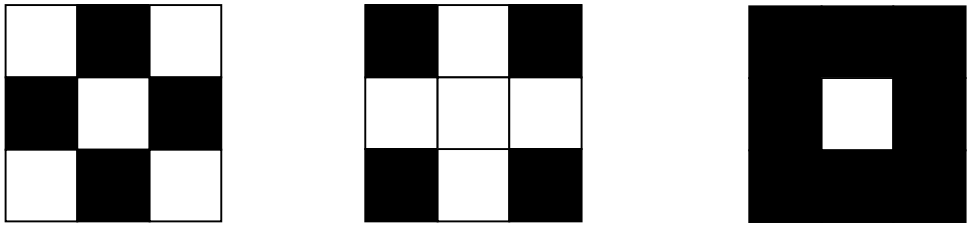
$$S_2 = \sum_{i=1}^{N} (w_{i\bullet} + w_{\bullet i})^2$$

4.4 Neighborhood

The concept of neighborhood is crucial to the determination of spatial autocorrelation. The choice of the type of neighborhood depends on the relationship under examination. In this research, the focus is on the spatial relation between network worm incidents in a geographically close LAN. Neighborhood relations are determined for the geographical and topological cases.

4.4.1 Geographical

In traditional plant epidemiology, when grid spaces are under consideration, movements of different pieces of the game of chess are used to define neighborhood characteristics. Figure 9 shows the different kinds of neighborhoods that exist at the first spatial lag for the center square. The black squares indicate neighbors. Therefore, Figure 10a displays the rooks neighborhood; Figure 10b shows the bishop neighborhood; Figure 10c shows the queens neighborhood.



a b c

Figure 9. Neighborhood definitions a) rook neighborhood, b) bishop neighborhood, and c) queens neighborhood

The same movements can be used to explain the neighborhoods at several distances. Figure 10 shows the rooks neighborhood at spatial lag 2 for the center square.

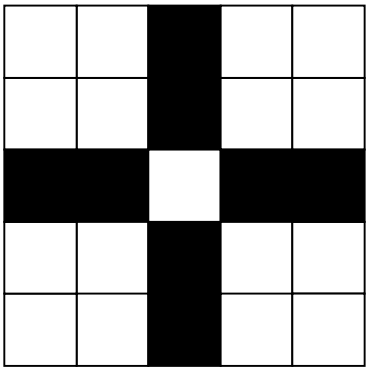


Figure 10. Rooks neighborhood with spatial lag 2

A 1:50 scale map of Virginia Tech was divided into an 18 x 18 grid structure. A 1:100 scaled map of Virginia Tech without the grid partitions is shown in Figure 11. A fragmented 1:50 scaled map of Virginia Tech can be obtained from the Virginia Tech Architects website at (http://www.unirel.vt.edu). The grid spaces were approximated via an overlaid standard 8 x 8 graphing sheet. Two squares on the sheet were joined such that each grid square eventually was a 16cm x 16cm square on the sheet. Each grid square contained approximately one building on campus.

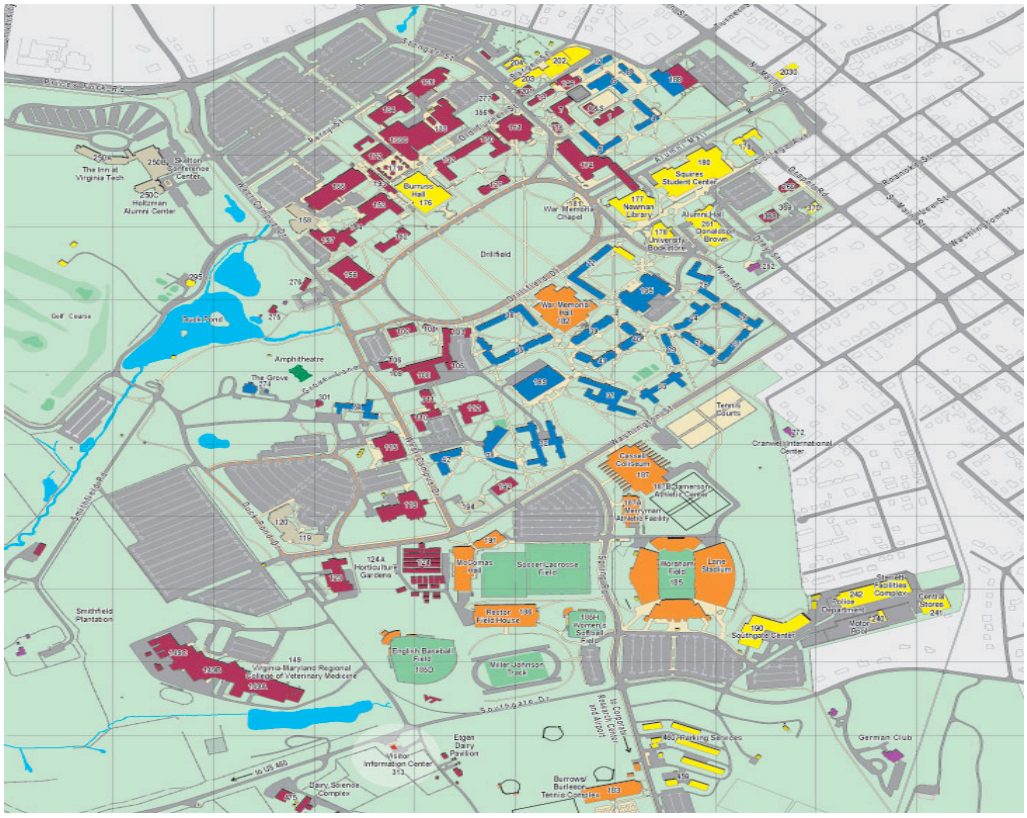


Figure 11. 1:100 scaled map of Virginia Tech

To maximize the analysis of influence but still maintain a statistically computable number of combinations, the queens neighborhood structure with only a first order spatial lag only was considered for our analysis.

In addition, certain neighbors might not exist at all spatially within this neighborhood structure. Absence of computers in a certain portion of the grid, such as a lake or building with no computers (greenhouses) resulted in the reduction of the weight of the non-existent neighbor to 0. Non-existent neighbors would not have any effect on the number of infections within another grid square. In addition, the effect one neighbor can have on another neighbor does not increase when a neighbor is unable to affect the infections. Therefore, their weight was maintained at a constant 1 value.

Consequently, a first order neighbor if it had computers was assigned a weight of 1, otherwise it was assigned a weight of 0. This process was repeated for each grid square, thus producing the weight matrix for the geographical case.

4.4.2 Topological

The topological case is quite different and more complicated than the geographical case. No natural divide is evident for recognizing if a computer is a neighbor of another computer and for assessing how much consideration should be given to this problem.

At the highest level, a computer topological network has the structure of a tree (Figure 12).

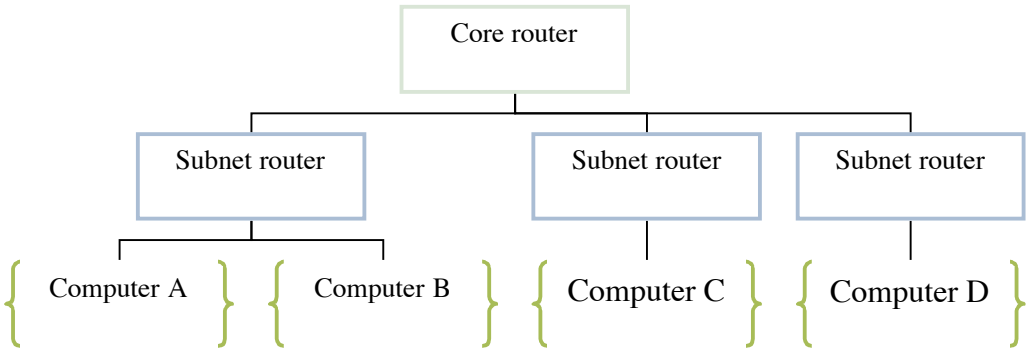


Figure 12. Tree hierarchies of computer networks

Figure 12 indicates that not every computer has neighbors at its level. This research only considered the class C level domain, so the task is somewhat simpler. However, the issue of levels and neighborhoods remains an equally complex problem as the subnet structure in Figure 13 shows.

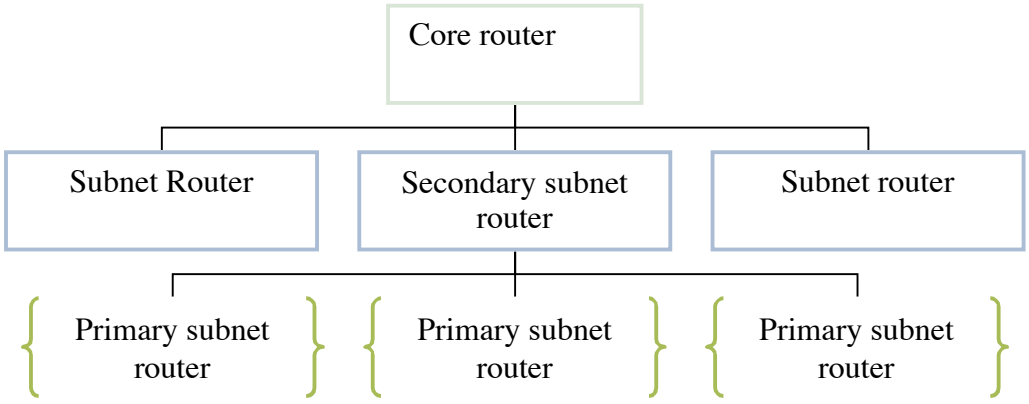


Figure 13. Subnet structure under a core router

Once the number of subnets begins increasing dramatically, redundancy is introduced into network paths. This redundancy helps networks stay active, even if one particular route is full or one section of the network is down.

Redundancy in network topological structures can be modified by various means. An example of a redundant structure is the star network topology shown in Figure 14. This structure produces multiple ways to communicate from one subnet to another. However, in practice, computer network architects use a combination of different topologies that they have developed from their intimate knowledge of the network.

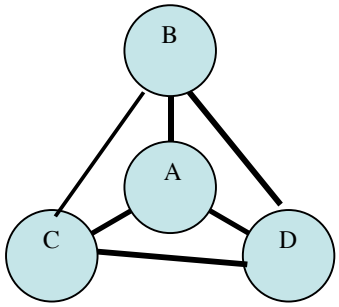


Figure 14. Star network topology for subnets A, B, C, and D

The weight of the edge between two subnets is determined by the probability of an infection from a computer in one of the subnets under consideration following a network topological route to a computer in the other subnet under consideration. The neighborhood weight, therefore, primarily depends on the subnet connections of the network topology. Figure 15 demonstrates the calculations of neighborhood weight.

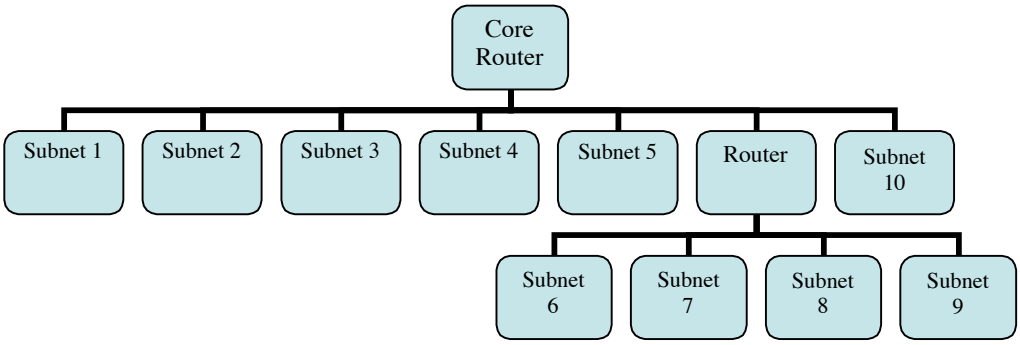


Figure 15. Example subnet structure

The probability that subnet 4 will choose subnet 5 over all its neighbors is 1/7. This calculation comes from the seven different paths

that the infection can take on an outward bound path from subnet 4. Therefore in the weight matrix row for subnet 4, the weight for subnet 5 is set as 1/7. However, if a subnet is not in the same child of the same parent, then end-to-end probabilities are used. To determine the weight of the neighborhood between subnet 4 and subnet 9 (Figure 16), the probabilities of direct neighbors from 4 to the parent of 9 is 1/7. The probability that the choice at this point will be subnet 9 is 1/4. Therefore, the total probability is 1/7 x 1/4 = 1/28. Finally, all the weights for the subnet are normalized so that the weight of each individual subnet is normalized to 1. In effect, all the probabilities are summed up and then each probability is divided by the sum of all probabilities to produce the weight.

Such a strategy leads to an exponential weighting scheme as the distance changes. To simplify the weighting scheme, only subnets under one core router are considered to be neighbors. The actual topological structure beyond the core routers is abundant with redundancies. Also, the probability values become extremely small (<0.0001) as the neighborhood definition stretches beyond the core router. This decrease in probability values is due to the large number of subnets that are encountered at every stage in the topography. In this research, neighborhood was defined as every subnet being considered a neighbor of the other as long as each subnet lies under the same core router within Virginia Tech's LAN.

This topological weighting scheme categorizes numerous subnets as a neighbor of one subnet, albeit at different levels. This weighting scheme is in stark contrast to the weighting scheme used for the geographical case. However, this weighting scheme better reflects the relationship between two subnets under the same core router.

5 Data Analysis

Already discussed are the choice of spatial autocorrelation and the different ways to calculate it. In the following section, discussion concerns the methodology behind the data collection, analytic methods, and the tools including their development, implementation and testing. Finally, the results are outlined and their significance is discussed.

5.1 Tools

Three distinct categories of tools were used for data analysis:

- Data storage tools— used for data storage and processing
- Analytic tools— used for calculation of input required for spatial autocorrelation, and
- Statistical tools— enables the actual spatial autocorrelation calculations

5.1.1 Data Storage Tools

A large data collection effort requires a substantial database for storing and querying the data. MySQL [2001] was chosen as the database for two primary reasons:

(1) MySQL is a stable database with strong reliability and performance characteristics,

(2) Dshield, the data provider, uses MySQL to maintain its records

The 4.0.14 version of MySQL was implemented on Linux i386 systems with a 2.4 kernel. This database primarily held all the data that was harvested from the Dshield database.

A day's record from the Dshield database consists of approximately 20 to 30 million individual firewall log entries. Records are stored in a MySQL database. A series of SQL command such as one in Figure 16 are used to collect pertinent information about the source address, target address, target port, date, and the time at which the intrusion was attempted. One such SQL command is shown below

```
Mysql> Select source, sourceport, target, targetport, date, time from
reports_731805 where (source like '128.173.%' and targetport =
'1434');
```

Figure 16. Sample SQL Query

Information collected in the above manner is stored in another table in the MySQL database. Other analytic tools use the data stored by the above methods for calculations of spatial autocorrelation.

5.1.2 Analysis Tools

Calculation of spatial autocorrelation requires two primary sets of values: the actual number of incidences in each grid or subnet and the neighborhood relationship between the individual grids or subnets. The analytic tools are used in calculations of these sets of values.

The neighborhood relationship between the entity spaces is stored in a 2-dimensional matrix. However, the number of incidences in each entity space is stored as a list in a file.

5.1.2.1 Neighbor Matrix Development Tool

In order to obtain statistical values for spatial autocorrelation, a matrix that details the relationship of the neighborhood is required. A tool was developed using gcc 3.1 and tested on Open BSD using Darwin. This tool interacts with a MySQL database using MySQL's C Application Protocol Interface (API). The database stores information on the location of an entity space with regards to the entire space. The basic output of this part of the program is a text file containing the relationships in matrix form.

The program must first be compiled to form an executable. The tool can be executed from the Unix command line by supplying two arguments after the executable name. The first argument indicates the kind of analysis to be performed, geographical (geo) or topological (topo). The second argument is the name of a file containing the grid spaces without a computer in them. An example of a grid space with no computers is a pond or simply a building like a greenhouse. The file supplied in the second argument speeds up the process by skipping computations for the grid spaces in the file. This file is especially helpful if there is large grid space or a large topological network.

Output can be redirected to a file from the command line. The source code for the tool is presented in Appendix A.

The algorithm for calculating the neighborhood matrix differs for the geographical and topological cases. Calculations for each of these cases are detailed below.

5.1.2.1.1 Geographical Case

The neighborhood structure assumes all neighbors of the grid square (considering queens neighborhood) to have a value of 1, except if the grid square is one of the exceptions noted above. Grid squares without a 1 have no influence on the possibility of contracting a virus. The names of those grid squares are stored in the file that is supplied as the second argument during the execution of the tool. If an empty file were provided as the second argument during the execution of the tool, the conclusion is that no grid space is empty. An example of running this case from the command line is shown below.

```
Shell>./executable_name  geo  file_with_nonexistent_grids.txt
```

5.1.2.1.2 Topological Case

The program internally constructs a tree structure of the neighborhood based upon the location information provided in the MySQL database. After forming the tree structure, the neighborhood weight matrix is calculated using the principles outlined in Section 4.4.2. The weight calculations are based on a probabilistic measure of the topographical path followed by an infection from one subnet under consideration to another. These calculations are only performed if both the subnets under consideration exist below the same core router.

In the topological case, the command line execution is similar to the one in the geographical case. An example of running the executable from the command line is shown below.

```
Shell>./executable_name topo   file_with_nonexistent_subnets.txt
```

The second argument provided while executing the program code is a file containing all non-existent subnets. This file can be an empty file. However, a file containing the unavailable subnets can help reduce the run-time of the program. Typically, run-time for the topological case is longer than that for the geographical case. Run-times were consistently recorded around the 20-second time range on a Power PC 800 system with 256 MB of RAM.

5.1.2.2 Incident Aggregation Tool

The incident aggregation tool determines the number of infections in each geographical grid space or topological subnet. The infection values are necessary for determining spatial autocorrelation.

This tool aggregates log records of incidents that are stored in the MySQL database tables by the SQL queries of the data storage tools. If a geographical analysis of the number of incidents is to be performed, the ip2vtgrid.php script is used. On the other hand, the ip2vttopo.php script is used for a topological analysis.

The output of the tool is a list containing an entity space with the corresponding number of incidents on each separate line. In this analysis, the entity space was a grid square for the geographical space and a subnet for the topological case.

The incident aggregation tool was developed in PHP (see Appendix B). This tool has been tested on a X86 machine running Linux with 2.4 kernel and PHP 4.2. The tool for the geographical case can be run from the command line as

```
Shell> php   ip2vtgrid.php
```

For the topological case, the tool can be run from the command line as

```
Shell> php   ip2vttopo.php
```

The output can also be redirected to a file from the command line.

5.1.3 Statistical tools

The final group of tools uses a statistical program SAS for the actual calculation of spatial autocorrelation. This research used SAS 8.02 running on a Windows XP platform.

Any statistical package that has the capability of calculating spatial autocorrelation could be used for this purpose. The choice of SAS was simply based on resources available as part of the Virginia Tech community.

Spatial autocorrelation is calculated using SAS macros that are programmed to use the neighborhood weight matrix and the incident list as input. The output of the SAS macro is the value of Moran's I spatial autocorrelation with its associated *p*-values. The *p*-values indicate the level of confidence in the Moran's I value obtained.

The SAS macros for the geographical and topological calculations of Moran's I are provided in Appendices C.1 and C.2, respectively.

5.2 Data Collection

Data was collected at the SANS Internet Storm Center. Data collection was facilitated by Johannes Ullrich at the SANS Internet Storm Center. The Internet Storm Center backs up data occasionally and stores information for interesting periods of worm activity. For this research, data was extracted for the Linux Slapper, SQL Slammer, and MS Blaster network worms (Table 2).

	Start Date	End Date
Linux Slapper	09/15/2002	09/30/2002
SQL Slammer	01/22/2003	02/10/2003
MS Blaster	08/15/2003	08/31/2003

Table 2. Time periods for data collection

Traffic patterns that match these worms can still be found occasionally on the Internet. However, the main phase of any infection is during the early hours of the worm spread. This main phase is illustrated by the exponential rise of the total number of computers infected by the worms (Figure 2 [on page 20]).

The Dshield database collects logs and processes them on an hourly basis. Participants may choose to submit logs, at weekly, daily, or even hourly intervals. To collect as many logs as possible for the relevant periods, log data must be collected for at least 15 days after a worm starts spreading.

Since people from all over the world submit these logs, the existence of an infected computer on Virginia Tech's network can be

determined by two conditions. The first condition is that an infected computer from the Virginia Tech network has to attempt an infection to a computer having a firewall with logging capabilities. The second condition is that the person owning the machine has to submit the firewall logs to Dshield.

Network worms tend to infect computers by selecting random IP addresses and attempting to infect them across the Internet. The spread of a successful worm has to be rapid or network administrators will set up blocks at the perimeter. The rapid spread attempt also ensures that as many computers as possible will be infected in the shortest time possible. For example, the SQL Slammer worm of January 2003 spread across the Internet in approximately 30 minutes. This rapidity results in the worm scanning several IP addresses in a short time. Since these IP addresses are randomly generated, the worm has equal probability of infecting a computer next to it and one across the world. An infected computer will, thus, have attempted to infect computers all over the world with equal probability.

Approximately 200 million records are added to the Dshield database each week. The possibility is high that an individual who sends a log will have recorded an intrusion attempt. Although this procedure is not the most foolproof method of detecting all infected computers, most of them can be detected in this way.

This research is centered on three major network worms that used a random IP generator to spread from one computer to the next. The time periods for the spread of the worms are detailed in Table 2. Moore and Shannon [2001] and Zou *et al.* [2003] have shown that the time series spread of a worm follows an exponentially increasing curve. The network worm spreads to almost all susceptible computers in the first 2 days after the worm is released. Consequently, analyzing

the data from 15 days will provide all the computers affected by the virus.

This section has outlined data collection methods. However, the best method for collecting data for a LAN is by deploying an IP packet-sniffing engine on the core router that links the LAN to the Internet. Unfortunately, such data are often not stored and, hence, are unavailable. Dshield or a similar log collection source can be an important resource in this respect.

5.3 Data Analysis

The sample data collected from the sources mentioned above were analyzed to form a simple list of values using the analytic tools described in Section 5.1. This list contained the number of incidences of the virus in each area under consideration. Only the grid areas in which an infection was possible (i.e. computers were available) were considered. This eliminated all areas such as parking lots and lakes from the data in the geographical case. In the topological case, subnets that did not exist on the network were eliminated from the list.

The incidents in an area were standardized to ensure the principle of statistical stationarity. In simple terms, we were simply ensuring that an entity space did not have a higher number of occurrences simply because it had a higher number of computers. The incident aggregation analytic tools in Section 5.2.1.2 produced a list that indicated the percentage of computers in a location that were infected by the virus. This crucial step was necessary to fulfill the statistical stationarity principle.

The analytic tools also produced the weighting matrices described in Section 4.3. This weighting matrix and the data obtained in the earlier step were used as input to the SAS program. The SAS Moran's I macro calculated the spatial autocorrelation (I) and significance values (*p*) for the virus data provided.

The Moran's I value indicated the extent of spatial autocorrelation. The significance of the spatial autocorrelation was indicated by the *p*-values produced by the macro. A value below 0.05 was considered fairly significant and acceptable in this research.

Finally an agreement test was conducted between the three virus incidences to determine if the spread was in agreement. This agreement test verified that the spatial autocorrelation (I) values were not occurring by chance. Since the starting points of all three epidemics within the Virginia Tech network were different, the agreement test verified that the spatial autocorrelation value remained the same regardless of where the epidemic started within a LAN. In real-world examples, these values may not be exactly the same. However, the agreement test verifies if the values are similar.

The expected value of I [Cliff and Ord, 1981] is given as

$$E(I) = \frac{-1}{(N-1)}$$

where *N* is the number of locations under consideration.

In the geographic case, the worm could possibly have infected 108 grid squares. Therefore, E(I) = - 0.0094. Similarly, in the topological case, the worm could have infected 146 subnets. Therefore, E(I) = -0.0069.

5.4 Results

The results are given here in tabular format for easier interpretation.

Topological case				
	Number of subnets = 146			
	E(I) = -0.0069			
Worm under analysis	Moran's I value	*p*-value	Variance	Starting point (Class C subnet)
MS Blaster	-0.0194	0.9530	0.0000561	37
SQL Slammer	0.0093	<0.0001	0.0000038	105
Linux Slapper	0.0671	<0.0001	0.0000006	149

Table 3. Spatial autocorrelation results for the topological case

Geographical case				
	Number of grids = 108			
	E(I) = -0.0094			
Worm under analysis	Moran's I value	*p*-value	Variance	Starting point (grid square)
MS Blaster	0.1120	<0.0001	0.0001931	P8
SQL Slammer	0.1449	<0.0001	0.0000025	D15
Linux Slapper	0.0329	<0.0001	0.0000006	H14

Table 4. Spatial autocorrelation results for the geographical case

As shown, the *p*-value of the Blaster worm under topological analysis is very high. This *p*-value of the Blaster worm indicates that it spreads independently of space. Therefore, the Blaster worm was not

compared to the other two in the topological case. The remaining results of the agreement test are shown below.

Topological case		
Comparisons		*p*-value
Worm	Worm	
Slammer	Slapper	0.6547

Table 5. Agreement test for topological case

Geographical case		
Comparisons		*p*-value
Worm	Worm	
Blaster	Slammer	0.0578
Blaster	Slapper	0.0009
Slammer	Slapper	0.0253

Table 6. Agreement test for the geographical case

5.5 Analysis of Results

Analysis of the results obtained from the data analysis is explained in this section. The two cases are analyzed separately.

5.5.1 Topological

The *p*-values are high for the Blaster worm (Table 3). Thus, the topographical case does not show an autocorrelation between the incidents of the Blaster epidemic in the LAN. The corresponding *p*-values for the Slammer (<0.001) and Slapper (<0.001) worms indicate that spatial autocorrelation exists. Therefore, a spatial dependency exists in the spread of the Slammer and Slapper worms within the network topography of the Virginia Tech LAN.

The agreement test was used to determine if the spatial autocorrelation values of the two worms under examination were related. In effect, the test determines if the worms spread in a similar way. The agreement test is only conducted if the worm itself spread in a spatially dependent way.

In the topological case, an agreement test was performed between Slammer and Slapper results. The null hypothesis of an agreement test is that the values are not in agreement. In this research the null hypothesis is that the spatial spreads were not similar. A high *p*-value indicates that the null hypothesis is true.

The *p*-value in Table 5 is 0.654. Such a high *p*- value indicates that the null hypothesis is true. This means that the spatial spreads of Slammer and Slapper worms in the topological case are not the same.

5.5.2 Geographical

The p-value is low (~< 0.05) for each of the virus cases (Table 4). Each virus case displayed positive spatial autocorrelation, that is, neighboring grid spaces had similar values. This finding indicates that the spread of the virus was spatially dependent in all three cases.

An agreement test was then conducted between the spatial autocorrelation values. The null hypothesis for the agreement test is that the values were not in agreement. A high *p*-value (>0.1) indicates that the agreement test hypothesis is correct. The p-values of the agreement test are low (Table 6). Therefore, the null hypothesis was rejected. Hence an agreement existed between the spatial autocorrelation values. Therefore, the spread patterns of the three viruses in geographical space were the same.

6 MIDS

In order to leverage the results of this spatial autocorrelation research, the design of an IDS called the Middle-range Intrusion Detection System (MIDS) is outlined. An important point to note is that such a system requires earlier occurrences of similar network worm events to determine the spatial autocorrelation present. Therefore, the first step involves implementing a DIDS on the network.

In the first stage, a DIDS was obtained from the SANS Internet Storm Center. Changes were made to the software so it would work on lower end capability systems. System requirements were gathered and the software was implemented on the Virginia Tech network. System administrators were asked to send firewall logs to Virginia Tech's DIDS [Marchany, 2003] A visualization component displayed attacks that were recorded on systems across campus. Figure 17 shows the visual component of the current system. A map of campus is displayed, along with the locations of the vulnerabilities occurring across campus. A legend on the left side of the figure shows attacks that are occurring. Such a component helps system administrators identify trends in attack patterns on the internal network so that they can modify their firewall or IDS rules. During this period, records of worm activity are captured and stored for analysis. This strategy gathers information on the subnets under attack and the number of computers that were infected.

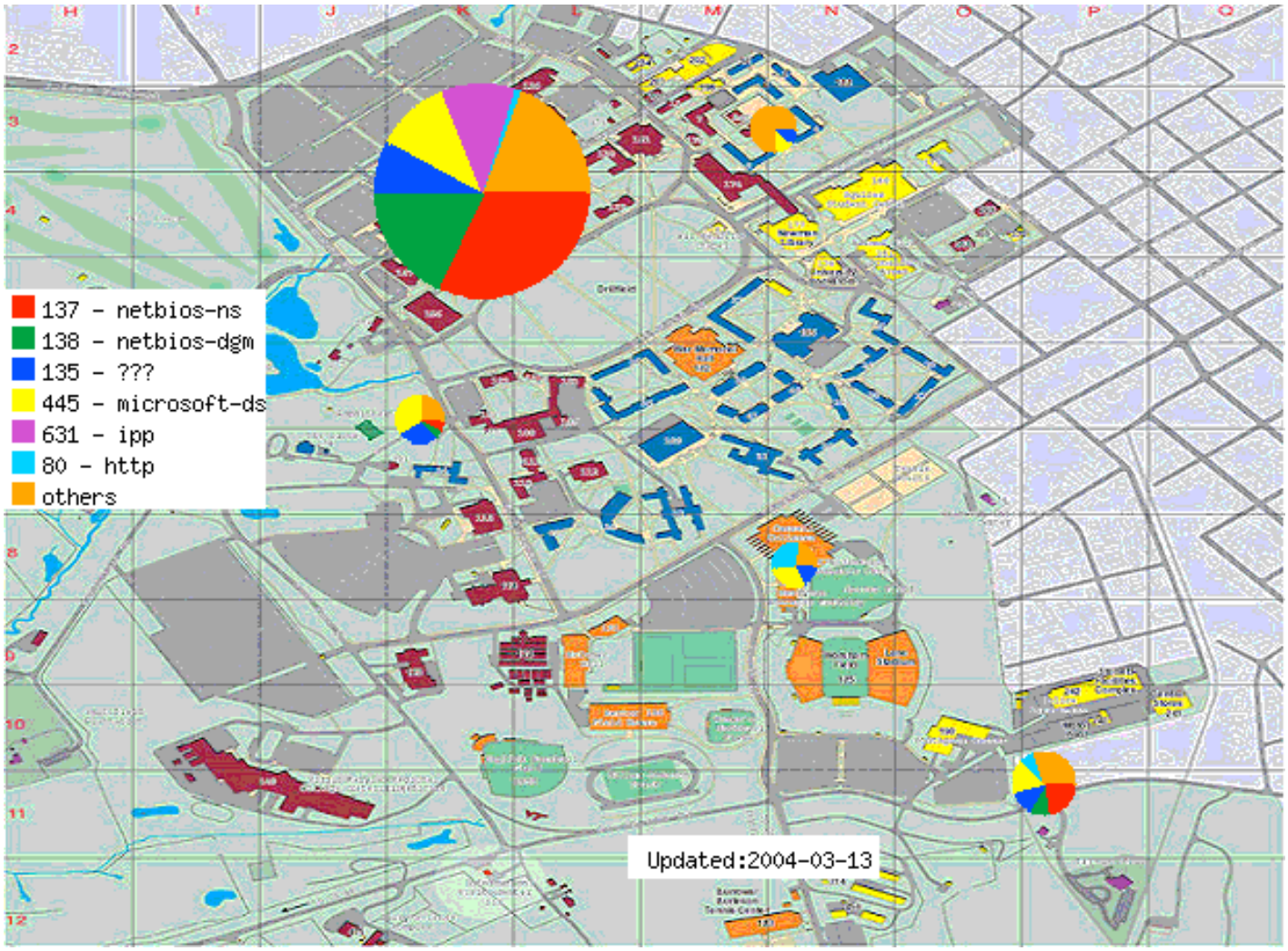


Figure 17. Virginia Tech local Dshield user interface

6.1 Design

Earlier discussion showed how the value of spatial autocorrelation can be used to help predict the spread of a network worm. The design of the system is outlined in this section.

During development, efforts were focused on keeping the product simple, minimalist and easy to implement. The MIDS architecture reflects this philosophy. The MIDS architecture design is similar to the D-shield DIDS architecture primarily because of implementation and cross-functionality reasons. The architecture (Figure 18) consists primarily of five components

- Agents
- Data aggregator
- Analyzer
- Spatial flow analyzer and predictor, and a
- Notification Component

Each of these components and their functionality has been explained below.

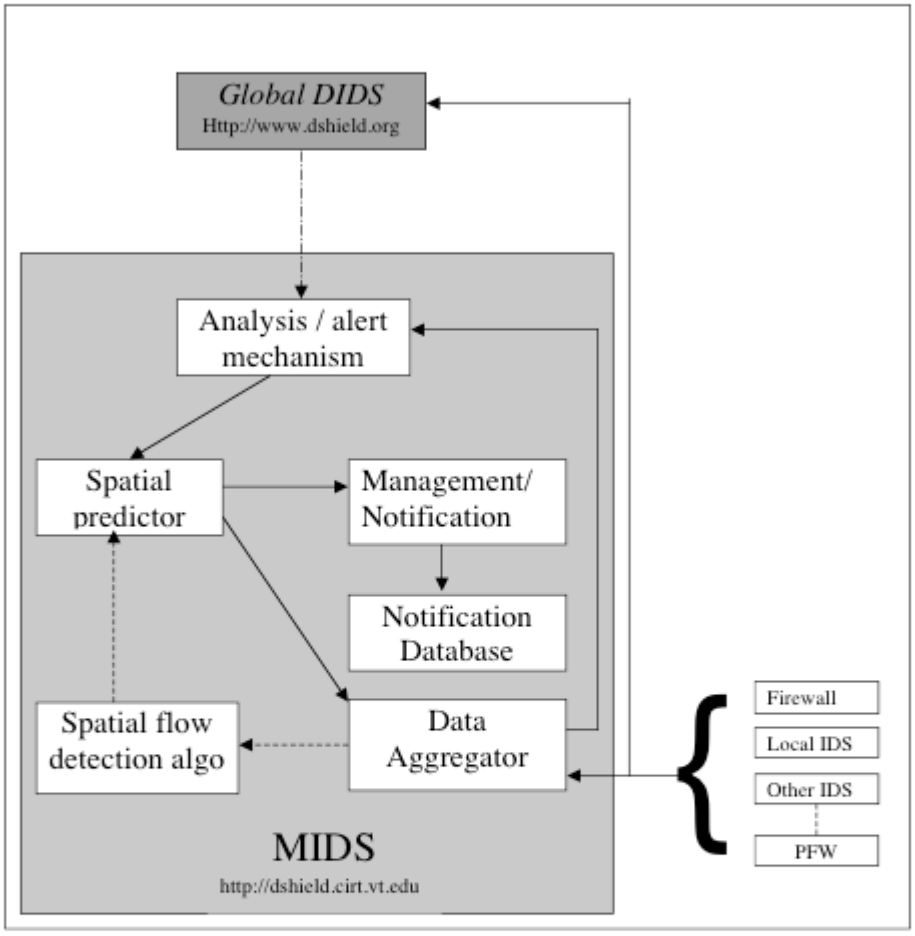


Figure 18. Architecture of MIDS

6.1.1 Agents

Agents are the primary and secondary data sources that provide information to the MIDS. Primary data sources are individual firewall logs, host-based IDS logs such as BlackIce or a packet analyzer such as Zone Alarm. Secondary data sources are network-based IDS such as Snort or router logs. Both of these sources can provide their logs to the MIDS.

Agents simply download clients that submit logs to the system and can be programmed to send its logs regularly to our system. This task can be achieved, for example, by classifying it as a cron job on Unix based systems. Several clients are available for different types of firewalls and IDS.

As MIDS is currently being developed for a Class A or Class B network level, the main agents are personal firewalls, perimeter firewalls and subnet IDS. The end user may sanitize data to protect privacy. No additional data scrubbing is done in the MIDS. However, care is taken not to expose end user reports as these can give an indication of the configurations of the end-user firewall and IDS systems.

Care is taken at the data aggregator level to delete duplicate records so that duplication does not affect our analyses. Duplicate records are identified by multiple fields (source, destination, timestamp, etc.) before they are deleted. A notice is sent to the user that the record has been formatted and added to the database. In case of problems, notices are also sent so that these can be solved in the future.

Agents are often deployed at multiple levels because packet dropping may be enabled at different levels in the network hierarchy.

The decision to drop the packet versus simply logging it is made in the context of the risk of the intrusion and the possibility of the service existing at a lower level in the network hierarchy.

6.1.2 Data aggregator

The data aggregator component is a combination of a MySQL database and PERL/PHP scripts. Agents send their data in text file format to the MIDS. These files are stored in a folder on the server for a time period specified in the configuration files. The data aggregator runs periodically, depending on the system resources available and the need for immediate updates.

On the current system, data aggregation after 8 hours is a reasonable period of time before an update begins. However, this time balance was identified after a substantial trial-and-error process. System administrators will need to adjust this time period based on the number of submissions and system resources available. Typically, for a system with submission logs from a Class B network, 4 hours are needed to run the updates if the updates are run after 8 hours. However, if the updates are run hourly, they tend to overlap in processing times.

The scripts contained in the data aggregator module have error checking, bounding, and formatting mechanisms to convert the raw files to database storage files. Checks are also performed at this stage for any duplicate records. This check is important to maintain the validity of the results. Information is examined for removal of any indication of the identity of the original submitter, and a notice is sent to the submitter of the action taken.

The data aggregator also controls the database mechanism and is the only module that is allowed to make changes to the database. Hence, appropriate security considerations should be put in place. These security precautions are necessary to maintain the privacy of

the submitters. Log records can indicate the configuration of firewalls or routers on the network of the submitters. This lapse in security could potentially lead to the attackers circumventing the security measures in place on the network of the submitters. A packet sniffer on the network containing the data aggregator can flag any suspicious packets and generate a note to the system administrators. In addition, care is taken to assure that none of these services are available from the Internet.

The data aggregator modules, the database itself and the log storage directories are on the same physical machines. This architecture is used to speed up the aggregation process. If a high data network bandwidth is available, the log storage directories can be separated from the rest of the aggregator. Such architecture can help increase security of the database, but negatively impacts performance (as mentioned earlier).

The size of the database has to be constantly monitored. A noisy worm can cause high amounts of logs to be submitted and, thus, consume disk space. While disk space is not a terribly costly entity, choices regarding the amount of space that may be used should be made before system deployment. The data aggregator stores logs for 60 days. Extending this functionality, however, will not be a major development challenge.

Also, special scripts have been developed for storing data for important events, such as a new virus. In particular, such data is stored separately and for longer periods of time. Such utility scripts and databases assist us in determining of spatial autocorrelation, which will be valuable in the ultimate usability of the system.

6.1.3 Analyzer

As explained earlier, most software was derived from the DIDS obtained from Dshield. Therefore, the architecture bears a resemblance to the Dshield DIDS architecture. The analyzer component is analogous to the central analysis server of Dshield. Modifications have been made to several components to improve stability and optimize the system in a smaller network architecture.

One of the major components added to the analyzer is the alert mechanism. Either the local analyzer or a global DIDS mechanism can trigger this alert mechanism. In this way, both local and global emergencies can be captured.

Future ideas include a structure with three separate alert levels. The lowest alert level would be one where the local analyzer detects a series of anomalies in the internal network and indicates this detection as an emergency to the spatial predictor. A second level of alert would be one where the locally detected anomaly is passed to the spatial predictor; the global DIDS indicates the presence of the same kind of traffic after a reasonable amount of time. The highest level of alert would be triggered when the local analyzer has yet to detect an anomaly, but the global DIDS indicates the presence of a global intrusion. This alert would mean that data is insufficient for the local analyzer to detect a problem with the network, but a global scale warning is occurring (Figure 18).

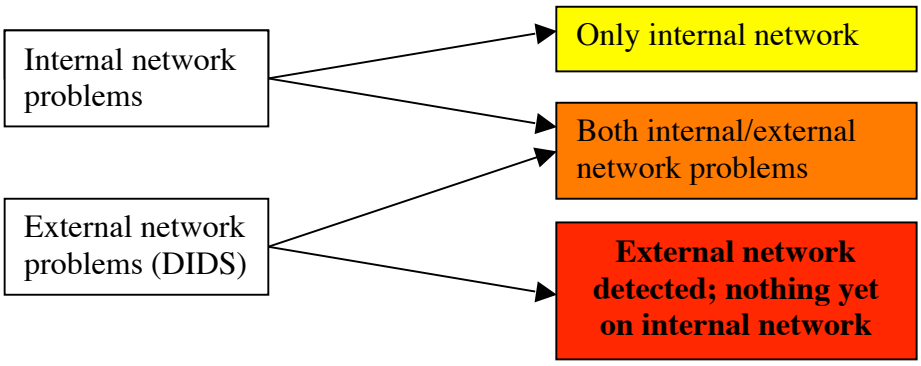


Figure 19. Threat level determination

If the spatial prediction mechanism can predict the path that computer viruses will follow at this stage, network and system administrators can be warned immediately.

The three levels of warning can also dictate the resources that must be allocated to the intrusion analysis process. This logic must be embedded in the analyzer component. Although a very simple gradation of levels of severity is presented here, this scheme has potential to develop into a massive, artificial intelligence problem, which this researcher would like to avoid—at least for now.

6.1.4 Spatial analyzer/predictor

The spatial analyzer/predictor module is the core motivation and the major advantage of MIDS over DIDS. Two components make up this module, the spatial analysis component and the spatial prediction component. These two components have been separated in the architecture diagram since they are separate parts of the system. However, one single thread links both parts, i.e., spatial analysis.

The spatial analysis component examines the data stored in the data aggregator for different events. All current data are evaluated to determine if the trend of an intrusion has an epidemic/pandemic quality. Such a trend is indicated by a sudden surge in the number of events recorded for a particular port or activity similar to the one shown in Figure 1. If such a sudden surge is observed, the MIDS internally classifies this event as an epidemic.

An attempt is then initiated to find a relation between virus incidence and the geography/topology of the LAN. Previously calculated spatial autocorrelation values will help detect the probability of neighbors receiving a worm. At this stage, the subnets where the epidemic exists have been identified.

Knowing the spatial autocorrelation value can assist in determining the parts of the geographical network topology where the virus will spread. Different types of intrusions may have different values of spatial autocorrelation. This research has only concentrated on one specific type of intrusion, the network worm.

The end result of the spatial analysis component is a model with one variable, that is, the starting point of an epidemic. This model is called the spatial flow model. A special spatial flow detection algorithm

is used to identify the existence of a particular virus model, if several exist.

Once the appropriate model is determined, the spatial prediction component locates the current instance of intrusions on the LAN. When the spatial analyzer receives an alert from the data analyzer component, the spatial analyzer completes mathematical calculations to determine the flow pattern. This flow pattern determines the next subnet the worm may attack. The spatial analyzer component forwards this information to the notification component. The notification component can then determine the kind of notification that must be provided and to whom. The notification component is detailed further in the next section.

6.1.5 Notification Component

The notification component is at the end of the MIDS workflow architecture. It handles the task of deciding which notification method should be used, depending on previous responses, time of day, and severity of the threat.

The notification component interfaces with a database containing contact information for the network administrators of the various subnets. The primary methods of notification currently envisioned are e-mail, pager messages, and instant messages that can be used individually or in combination, depending on the severity of the situation. This module has been split into two components in the architectural diagram to emphasize the exact part with which the other modules interact.

The notification methods can depend on the level of emergency as determined by the data analyzer component. Several strategies of dynamic takeover of system resources have been suggested. However, system management must determine implementation of such strategies.

6.2 Example Usage Scenario

In this section, a usage scenario for the MIDS is described. To highlight the important steps that must be taken during set-up, the implementation process is also described.

An enterprise, ABC Corporation, decides to implement the MIDS for active detection and prediction of virus spread through the LAN. The first step is to analyze previous occurrences of virus on their LAN. For this purpose, they accumulate historical data about virus occurrences on the LAN from a global DIDS, such as Dshield. They determine the spatial autocorrelation of previous virus occurrences on the LAN and store this information in a database.

The MIDS system, described above, is installed after primary testing. System administrators can choose to show the display to internal staffers. Such a display can help identify the progression of intrusions across the LAN.

A new worm, New.Worm, begins spreading across the Internet at midnight. The global DIDS at Dshield detects this activity at 2 am. A message is sent that contains the virus name, the port number the virus uses to spread, and the speed of spread. This message is received by the alert mechanism at ABC.

The spatial analysis component searches for incidences on the LAN where a computer might be trying to spread the virus using the port number mentioned in the Dshield notification. An incidence is identified in the north corner of the ABC campus.

Using the predetermined spatial autocorrelation value, along with any corresponding model that may be stored, an analysis is

performed on the structure of the LAN. Depending on the prediction of the spread and the intensity of the worm, several system administrators in the northern area of the campus are paged immediately to react to and impede the flow of the virus in that quadrant. The remaining system administrators are notified of the situation via e-mail.

At approximately 5 am, the virus spread has begun to slow down globally. The Dshield message center updates its information and sends it out. Detecting this slow down, the MIDS alert mechanism at ABC Corporation updates the notification component to reduce the emergency level.

Several system administrators in the west and east quadrants of the campus, where the spatial spread was most probable, have been paged by this time. However, due to the lowering of the emergency status, the remaining system administrators are not paged at this time. However, an e-mail update notifies them of the reduction in intrusions.

Although the remaining system administrators have not been disturbed, a close watch on intrusions is maintained. In case of a change in status of the spread of the worm, the notification system can begin actively paging the remaining system administrators.

7 Conclusions and Future Work

7.1 Conclusions

This research addressed the spatial spread of computer network worms. Earlier research has focused primarily on the time-series spread of network worms. Time-series models have potential for predicting the total number of computers that a worm will infect. However, they cannot predict the next subnet or area that will come under attack.

For the purpose of tracking the spatial spread of worms, related research in similar fields was examined. Similarities between plant and computer epidemics were presented, along with the reasons for the choice of investigating plant epidemiology. Parameters such as clustering indexes and autocorrelation were discussed in the context of network worms. Spatial autocorrelation was chosen as the statistical property for investigation. The primary benefit of choosing spatial autocorrelation is that a single value is obtained that signifies if the values of occurrence of a virus in a certain place are indicative of its neighbor becoming infected. The objective was to determine if a relation among the spread of worms within a LAN exists. Methods from epidemiology were used for this purpose.

Analysis was conducted on three network worms: MS Blaster (August 2003), SQL Slammer (January 2003), and Linux Slapper (September 2002). Data was obtained from the Dshield DIDS. The spatial spread in the Virginia Tech LAN was analyzed in the geographical and topological contexts.

The geographical spread was analyzed by dividing a 1:50 scaled map of Virginia Tech into equal sized grids. Grid spaces where computers did not exist were eliminated from the analysis. All grid spaces sharing a corner or a side with the grid cell under consideration were considered as neighbors.

The topological case was analyzed using the network topology of the LAN. The virus spread was considered at the Class C network level. All Class C networks under the same core router were considered as neighbors.

Data analysis was conducted using the data obtained from the Dshield database. The neighborhood structure was determined as mentioned earlier. Moran's I value of spatial autocorrelation was calculated for each case. An agreement test was conducted to test if the spatial spreads of the three viruses were similar.

The spread of the three viruses, considering the topology of the Virginia Tech network, indicated mixed results. While the spreads for Slammer and Slapper indicated a positive spatial autocorrelation, the Blaster worm spread was spatially independent. The agreement test for Slammer and Slapper showed that the spreads were not the same. This finding indicates that the spread of a network worm is not dependent on the topographical structure of the network.

The results of the geographic analysis indicated that the three viruses spread in a spatially dependent way. Further, the agreement test on the spatial autocorrelation values showed that the spread pattern of the three worms was similar in the geographical context. This finding indicates that if a new network worm with the same characteristics as those of the worms

examined were to start spreading across Virginia Tech's campus, a similar spatial autocorrelation value could be expected.

The geographic locations where a new network worm would spread might be extrapolated from these findings. This process would involve determining the geographic grid locations that the worm has already affected. Using the spatial autocorrelation value, some reverse engineering may be required. A network administration database can help identify the personnel responsible for managing the network. With this goal in mind, the design of the MIDS system was outlined. MIDS can issue warnings to network administrators about the spread of a worm to their networks.

Network worms spread via computer networks. However, their spread shows a geographical dependency. Consequently, this result was not intuitive and in fact, was surprising. A few conjectures may explain the reason for such a geographical dependency.

Conjecture 1:

> Humans tend to breed in similarity, i.e., similar individuals live in close geographical proximity. Due to this, trust relationships exist between geographically close computer networks. Since trust relationships are unchecked by firewalls, this trust enables a higher number of computers in the geographic vicinity to become infected. For example, most administrative buildings at Virginia Tech lie in the same geographic vicinity. A trust relationship exists between the networks of these buildings. Firewall rules allow these networks to communicate on unprotected ports. An infected computer in Virginia Tech's admissions office is more likely

to infect a computer in the president's office than a computer in a student's residence hall.

Also, the results show a surprising skew for the p-values of spatial autocorrelation in only one case – the topological analysis of blaster. The *p*-value for Blaster in the topological analysis is 0.95 (Table 4). Another conjecture may explain this occurrence.

Conjecture 2:

The level of security awareness among Virginia Tech's network administrators was extremely high when the Blaster worm started propagating. The reaction of the Virginia Tech networking group to the Blaster worm was immediate. All traffic leaving Virginia Tech's network over port 135 (the port that Blaster used to spread) was blocked at the Virginia Tech core router minutes after news of the new worm became available. The isolation caused by the network block affected the spread of the worm in the network topology.

7.2 Future Work

This research has demonstrated the possibility of detecting the spatial spread of a network worm. A sample field of study, plant epidemiology, can be used in analyses so that further models in this field can be directly or indirectly applied to computer epidemiology. Although spatial autocorrelation is indicated by this research to exist in the geographic context, other parameters may better indicate the spatial spread of network worms. In particular, SADIE clustering indexes [Perry, 2001] show promise in determining the areas most susceptible to a virus.

This research concentrated on network worms, a specific type of computer viruses. Further research would be helpful on different types of computer viruses.

The analysis of a network worm requires time, and a high amount of skill is needed to reverse engineer a computer virus. This research indicates a high consistency in the spatial autocorrelation value in the geographical spread of the worm. If a new virus shows a spatial autocorrelation value consistent with a certain kind of computer virus, then the characteristics of the worm may become immediately apparent. This data would save analysts from complex reverse engineering. These methods could even benefit reactive security methods so that holes in network defenses can be immediately plugged. However, further research is required to determine if these methods could be used for this purpose.

References

Airwise News (2003) "Worm affects Air Canada", August 20, 2003 < http://news.airwise.com/stories/2003/08/1061375870.html>

American Heritage Dictionary (2000), "Definition of epidemic", January 11,2004 < http://dictionary.reference.com/search?q=epidemic&r=67>

Anderson, R. and May, R. (1985) "Age related changes in the rate of disease transmission: implications for the design of vaccine programmes", Journal of Hygiene (Cambridge), 94:365-436, 1985

Anderson, R. and May, R. (1984) "Spatial temporal and genetic heterogeneity in host populations and the design of immunization programmes", In IMA journal of Mathematics applied in medicine and biology, 1984

Anselin, L. (1995) "Local indicators of spatial association - lisa," Geographical Analysis, 27:-115, 1995

Baldwin, Lawrence (2001) "MyNetWatchman - Vision", December 20, 2003 <http://www.mynetwatchman.com/vision.htm>

Balthrop, J., Forrest, S., Newman, M., and Williamson, M., (2004)"Technological Networks and spread of computer viruses", Science 23 April 2004: 527-529

Bautts, Tony (2003), Slow Down Internet Worms with Tarpits, Security Focus, < http://www.securityfocus.com/infocus/1723>, August 21, 2003

Bolker, B. (1997) "Heterogenity of mixing and spatio-temporal models: advances and open problems", In RSS Epidemics Workshop 1997

Botha, M. and Von Solms, R. (2003), "Utilizing fuzzy logic and trend analysis for effective intrusion detection", Computers and Security, Volume 22, Issue 5, July 2003

Broersma, M. (2003) "Slammer-the first Warhol worm?", CNET News, January 15, 2004 < http://news.com.com/2100-1001-983197.html>

CAIDA (2001), "Code Red Spread Animations", Animated GIF, September 23, 2001, < http://www.caida.org/analysis/security/code-red/newframes-small-log.gif>

Campbell, C. and Madden, L. "Introduction to Plant Disease Epidemiology", pp 161-201, John Wiley 1990

Center for Disease Control (2003a), "An Introduction to Epidemiology", Excite Classroom

Center for Disease Control (2003b), "Basic Information about SARS", January 16, 2004 < http://www.cdc.gov/ncidod/sars/factsheet.htm>

Cheswick, B., Kocher P., McGraw G.and Rubin A., (2003) "Bacon Ice Cream: The Best Mix of Proactive and Reactive Security", Networld + Interop Conference, April 27 –May 2, 2003

Chou, (1997) "Exploring Spatial Analysis in Geographic Information Systems" Santa Fe, OnWord Press 1997

Cliff, A.D. and Ord, J.K. (1981) "Spatial Processes: Models and Applications" Pion Press

Computer Security Institute (2003), 2003 CSI/FBI Computer Crime and Security Survey, <http://www.gocsi.com/press/20030528.jhtml>, Aug 5, 2003

Daley, D.J. and Gani, J. (1999), "Epidemic Modeling, An Introduction", Cambridge University Press, 1999

Diniz-Filio, J., De Campos Telles, M. (2002), '' Spatial Autocorrelation analysis and the identification of operational units in conservation of continuous population‘’, Conservation biology, August 2002, Volume 16, Issue 4, Page 924

Dshield (2001), "Dshield: About", December 21, 2003, <http://www.dshield.org/about >

Frauenthal, J. (1980) "Mathematical Modeling in Epidemiology", Springer-Verlag

Frederick, K. (2002), "Network Intrusion Detection Signatures- Part 3", January 18, 2004 <http://www.securityfocus.com/infocus/1544>

Garfinkel, S., Spafford, G. and Schwartz, A. (2003) "Secure Programming Techniques: Part 4", Practical Unix and Internet Security, 3rd edition, O'Rielly Publication

Geary, R. C. (1954). "The contiguity ratio and statistical mapping", Incorporated Statistician 5:115-145

Goodchild M. (1987), "A spatial analytical perspective as geographical information systems", International Journal of Geographical Information Systems1 (4): 327-34

Haining, R. (1990), "Spatial data analysis in the social and environmental sciences", Cambridge: Cambridge University Press

Inella, P. (2000), "The evolution of Intrusion Detection Systems", SecurityFocus, November 16, 2001, <http://www.securityfocus.com/infocus/1514>

Kephart, J. O., Chess D. M. and White S. R. (1993), "Computers and Epidemiology", In IEEE Spectrum, 1993

Kermack, W. and McKendrick, A. (1932) "A contribution to the mathematical theory of epidemics. Part II. The problem of endemicity", Proceedings of the Royal Society of London, 138:55-83, 1932

Korzyk, A. Sr. (1988) "A Forecasting Model For Internet Security Attacks," National Information System Security Conference, 1998

Kriedl, O. and Frazier, T. (2003) "Feedback control applied to survivability: a host-based autonomic defense system", IEEE transactions on Reliability, Vol 52, No. 3 2003

Lemos, R. (2004) "Seeds of Destruction", Cnet News, January 15, 2004

Liu, P. and Li, L. (2002) "A Game Theoretic Approach to Attack Prediction", Technical Report, PSU-S2-2002-001, Penn State Cyber Security Group, 2002

Marchany, R. (2003) "VT-Dshield site is online- need volunteers", Virginia Tech Techsupport listserv, < http://listserv.vt.edu/cgi-bin/wa.cgi?A2=ind030>

Matheron, G. (1963), "Principles of Geostatistics", Economic Geology, 58, 1246-66

Moore, D. and Shannon, C. (2001), "The spread of the Code-Red worm", CAIDA analysis, <http://www.caida.org/analysis/security/code-red/coderedv2_analysis>

Moore, D., Shannon, C., Voelker, G.M. and Savage, S. (2003) "Internet Quarantine: Requirements for Containing Self-Propagating Code", In IEEE INFOCOM, 2003

Moran, P. (1948), "The interpretation of statistical maps. Journal of the Royal Statistical Society", Series B, 10:243-251

Morris, M. (1994) "Epidemiology and Social Networks: modeling structured diffusion", In Wasserman, S. & Galaskiewicz, J., editors "Advances in Social Network Analysis: Research in the social and behavioral sciences", pg 26-52, Sage, 1994

MySQL (2001) "MySQL Database Server", < http://www.mysql.com/products/mysql/index.html>

Nojiri, D., Rowe, J. and Levitt, K. (2003) "Cooperative Response strategies for large-scale attack mitigation", DARPA Information Survivability Conference (2003)

Ord, J.K. and Getis, A. "Local Spatial Autocorrelation Statistics: distribution issues and an application", Geographical Analysis, 27:-306, 1995

Orman, H. (2003), "The Morris Worm: a fifteen year perspective", IEEE Security and Privacy, Volume 1, Issue 5, Sept - Oct 2003

Paquette, J. (2000) "A history of viruses", SecurityFocus, January 16, 2004 < http://www.securityfocus.com/infocus/1286>

Perry, J. (2001) "SADIE", SADIE home pages

Reuters (2004), "PC viruses spawn $55 billion loss in 2003", CNET News, January 16, 2004 < http://news.com.com/2100-7349-5142144.html?tag=cd_top>

Robbins, Royce, "Distributed Intrusion Detection Systems: An Introduction and Review", SANS Reading Room, GSEC Practical Assignment, 2002

Salkever, A. (2001), "Patches don't make a security blanket", BusinessWeek Online, January 19, 2004 <http://www.businessweek.com/bwdaily/dnflash/aug200

SANS Institute (2001a), "About SANS", December 26, 2003 <http://www.sans.org/aboutsans.php>

SANS Institute (2001b), "Internet Storm Center: About", December 20, 2003, <http://isc.incidents.org/about.html>

Schneier, B. (2003), "Did Blaster cause the blackout", ZDNet, January 12, 2004 < http://computercops.biz/article4427.html>

Silicon Defense, Inc. (2002) "Counter Malice- About", January 19, 2004 < http://www.silicondefense.com/products/countermalice/>

Snort Network Intrusion Detection System (2000), "About: Snort", January 12, 2004<http://www.snort.org>

Staniford, S. (2003) "Containment of scanning worms in enterprise networks", To appear in Journal of Computer Security

Staniford, S., Paxson, V. and Weaver, N. (2002), "How to 0wn the Internet in your spare time", 11th Usenix Security Symposium

Symantec corporation (2000a), "Reference: virus", January 11, 2004 < http://securityresponse.symantec.com/avcenter/refa.html - virus>

Symantec corporation (2000b), "Reference: worm", January 11, 2004 < http://securityresponse.symantec.com/avcenter/refa.html - worm>

Symantec Corporation (2002), "Linux. Slapper. Worm", January 10, 2004 < http://securityresponse.symantec.com/avcenter/venc/data/slapper.worm.html>

Symantec Corporation (2003), "W32. Blaster. Worm", January 10, 2004 < http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.worm.html>

Tobler, W. R. (1970) "A Computer Model Simulating Urban Growth in the Detroit Region" in Economic Geography, 46: 234-240

Travis G., Balas, E., Ripley, D. and Wallace, S. (2003), "Analysis of SQL Slammer worm and it's effects on Indiana University and related institutions", University of Indiana, Technical Report

Ullrich, J. (2000), "DSHIELD", January 6, 2004 ,<http://www.dshield.org>

Unix Manual Pages, "Rand ()"

Wassenar, T. and Blaser, M. (2002) "Contagion on the Internet: Letter to the editor", Emerging Infectious Diseases, Vol 8, March 2002

Williamson, M. (2002) "Throttling Viruses: Restricting Propagation to Defeat Mobile Malicious Code" In Annual Computer Security Applications Conference, 2002

Yurcik, W., Korzyk, A. and Loomis, D. (2000) "Predicting Internet Attacks: On Developing An Effective Measurement Methodology" 18th Annual International Communications Forecasting Conference (ICFC'02)

Zou, C.C., Gong W., and Towsley, D. (2002) "Code Red Worm Propagation Modeling and Analysis". 9th ACM Conference on Computer and Communication Security

Zou, C.C., Gong, W., and Towsley, D. (2003) "Worm Propagation Modeling and Analysis under Dynamic Quarantine Defense ". Workshop on Rapid Malcode (WORM'03) 2003

Appendix

Appendix A

Appendix A.1: main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include "checks.cpp"
#include "grid.cpp"
#include "db_test.cpp"
using namespace std;
//function defintions
bool printFirstLine(char, char, int, int, int[100][19]);

int main(int argc, char* argv[]){
    if(argc != 3){
        cout<< "thesis needs two argument to be run: geo or
topo"<<endl;
        exit(1);
    }

    // GEOGRAPHICAL CASE
    if (strcmp(argv[1], "geo") == 0){
     char temp1 = 'A';
     char temp2 = 'R';
     int temp3 = 18;
     int size = (int(temp2)-int(temp1))*temp3;
     int grids[100][19];

     Grid G;
        //print first line containing names of the grids
     bool check = printFirstLine(temp1, temp2, temp3, size,
grids);
        if (check == false)
        {
            valueCheck C;
            C.fatalError("printFirst line failed");
        }

        // get in input from the command line and store as
string
        FILE * inputfile;
        inputfile = fopen(argv[2], "r");
        char input[1000];
        fscanf(inputfile, "%s", input);
        fclose(inputfile);
```

```
        // for every grid square
        for (int i = int(temp1); i <= int(temp2); i++){
            for(int j = 1; j <=temp3; j++){
                //create copy of grids array to edit for
each grid
                int gridvalue[100][19];
                for (int grid1 =0; grid1<100;grid1++){
                    for(int grid2=0;grid2<19;grid2++){
                        gridvalue[grid1][grid2] =
grids[grid1][grid2];
                    }
                }
                char tempgrids[8][5];
                //find list of neighbors
                G.findNeighbors(i,j, tempgrids);
                for(int no = 0; no < 8; no++){
                    //add ; at the end of the actual string
so that no half matches are inadvertently found, i.e. A1
does not match A12
                    strcat(tempgrids[no], ";");
                    //check if neighbor is in input file;
if it is leave it at 0, else print 1
                    if(strstr(input, tempgrids[no]) ==
NULL){
                        char letter = tempgrids[no][0];
                        int number=0;
                        // atoi and direct casting did not
work. Therefore a hard cast is made

                        for (int num=1; tempgrids[no][num]
!= ';'; num++){
                            int tempnumber =
int(tempgrids[no][num]) -48;
                            number = (number*10) +
tempnumber;
                        }
                        gridvalue[(int)letter][number]=1;
                    }
                }
                //now print everything
                cout<<endl;
                cout<<char(i)<<j<<":";
                for (int no1=int(temp1);
no1<=int(temp2);no1++){
                    for (int no2 = 1; no2<= 18; no2 ++){
                        cout<<gridvalue[no1][no2]<<":";
                    }
```

```
                }
            }
            cout << endl;
        }
    }
    else{
        //TOPOLOGICAL CASE
        if (strcmp(argv[1], "topo") == 0){
            //do topological stuff
            int grids[255];
            cout<<" :";
            //first print first line and store all the
information in a grid
            for (int i =0; i<256;i++){
                grids[i] = 0;
                cout <<i<<":";
            }
            cout<<endl;

            //access database and find out what
            dbAccess dbQuery;
            dbQuery.connection();

            for(int i=0;i<255;i++){
                int retval = dbQuery.get_data(i);

                if( retval != 0){
                    cout<<"Problems in output"<<endl;
                    exit(1);
                }

            }
        }
        else {
        cout<< "argument can only be geo or topo"<<endl;
        exit(1);
        }
    }
    cout<<endl;
    return 0;
}

/**
* function to print the first line of o/p, list of all
possible nodes
**/
bool printFirstLine(char start, char end, int max, int
size, int grids[100][19]){
```

```
     valueCheck C;
     if (!C.is_char(start) || !C.is_char(end) ||
!C.is_int(max) || !C.is_int(size)){
          C.fatalError("printFirstLine needs characters to
be passed first");
     }
      cout <<":";

        for (int op= int(start); op<=int(end);op++){
          char temp[2];
          temp[0]=char(op);
          temp[1]='\0';
          //print header line
          for (int k=1;k<=max;k++){

                         char temp1[3];
               sprintf(temp1,"%d",k);
                         char abstemp[2]="\0";
                         int n;
                         for (n=0; temp[n] != '\0'; n++){
                             abstemp[n] = temp[n];
                         }
                         abstemp[++n] = '\0';
                         char* temp3 = strcat(abstemp,
temp1);
                         //cout<<"op"<<op<<"k"<<k<<endl;
                         grids[op][k] = 0;
               //cout<<grids[op][k]<<endl;
                         for(int count=0;temp3[count] !=
'\0';count++){
                    cout<<temp3[count];
               }
               cout<<":";
          }
     }
     cout<<endl;
        return true;
}
```

Appendix A.2: grid.cpp

```
/*
 *  grid.cpp
 *  thesis
 *
 *  Created by rishi pande on Fri Mar 05 2004.
 *
 */
#include <iostream>
#include <string>
//#include "checks.cpp"
using namespace std;

class Grid{
    private:
    //void callGridName(int, int, int, char[8][5]);
    //void formGridName(int, int, char[4]);
    //friend fatalError(char*);
        /*
    * function callGridName
    * Just a simple function to repeatedly call
formGridName for parameters given
    * If out of bounds check fails, the termpgrid is
populated with 0
    */
    void Grid::callGridName(int gridletter, int gridnumber,
int tempgridnumber, char tempgrid[8][5]){
     char neighbors[4];
        char start = 'A';
        char end = 'R';
         //bounds check with hard-coded values :(
        if ((gridletter >= int(start) && gridletter <=
int(end)) && (gridnumber>0 && gridnumber<19) &&
(tempgridnumber>=0 && tempgridnumber <8)){
            formGridName(gridletter, gridnumber,
neighbors);
            int no;
            for(no = 0; neighbors[no]!='\0'; no++){
                tempgrid[tempgridnumber][no] =
neighbors[no];
            }
            tempgrid[tempgridnumber][no] = '\0';
        }
        else {
            tempgrid[tempgridnumber][0] = '\0';
        }
```

```
    }

    /*
    * function formGridName
    * input    int i     ASCII value of grid letter
    *          int j     integer value of grid number
    *          char gridname[4]    string stroing the grid
name
    *
    * forms the gridname for the values supplied
    */
    void Grid::formGridName(int i, int j, char
gridname[4]){
        //form the gridname of the focus cell
     char letter[4];
        letter[0] = char(i);
     letter[1] = '\0';
        char number[3];
        sprintf(number,"%d",j);
        strcat(letter, number);
        int no;
        for ( no=0; letter[no]!= '\0'; no++){
            gridname[no] = letter[no];
        }
        gridname[no] = '\0';
    }




    public:
    //void findNeighbors(int, int, char[8][5]);
    char tempgrid[8][5];
    Grid(){}
    ~Grid(){}

    /*
    * Find's all the neighbors by queen's neighborhood
definition
    */
    void Grid::findNeighbors(int i, int j, char
tempgrid[8][5]){
     /*valueCheck C;
     if (!C.is_int(i) || !C.is_int(j)){
          C.fatalError("findNeighbors requires integer
parameters");
          exit(1);
     }*/
```

```
        //calculate all neighbors for queen's move
        //down row, same column
        int gridletter= i+1;
        int gridnumber= j;
        callGridName(gridletter, gridnumber, 0, tempgrid);

        //up row, same column
        gridletter = i-1;
        gridnumber = j;
        callGridName(gridletter, gridnumber, 1, tempgrid);

        //up row, left column
        gridletter = i-1;
        gridnumber = j-1;
        callGridName(gridletter, gridnumber, 2, tempgrid);

        //same row, left column
        gridletter = i;
        gridnumber = j-1;
        callGridName(gridletter, gridnumber, 3, tempgrid);

        //down row, left column
        gridletter = i+1;
        gridnumber = j-1;
        callGridName(gridletter, gridnumber, 4, tempgrid);

        //up row, right column
        gridletter = i-1;
        gridnumber = j+1;
        callGridName(gridletter, gridnumber, 5, tempgrid);

        //same row, right column
        gridletter = i;
        gridnumber = j+1;
        callGridName(gridletter, gridnumber, 6, tempgrid);

        //down row, right column
        gridletter = i+1;
        gridnumber = j+1;
        callGridName(gridletter, gridnumber, 7, tempgrid);
    }

};
```

Appendix A.3: db_test.cpp

```
#include <iostream>
#include <iomanip>
#include <string>

#include <mysql.h>
//#include <libmysqlcient.a>
using namespace std;

class dbAccess{
    private:
    MYSQL *cnx_init;
    MYSQL *cnx_db;
    MYSQL_RES *result_set;
    MYSQL_RES *subnet_result_set;
    MYSQL_ROW row;

    unsigned int ctr;


    public:
    dbAccess(){};
    ~dbAccess(){};

    //starts the connection to the mysql database
    void connection(){
        cnx_init = mysql_init(NULL);
     if (cnx_init == NULL){
          cout<<"problems in mysql_init"<<endl;
          exit(1);
     }
     cnx_db = mysql_real_connect ( cnx_init, "localhost",
"root",
                "samtron40BN", "building", 0, NULL, 0);
     if (cnx_db == NULL){
          cout<<"failure in connect"<<endl;
          exit(1);
     }
    }



    int get_data(int temp){

        unsigned long dec = ip2Dec(temp);
//        cout <<"dec"<<dec;
```

```
        char query_string[99] = "select region from
vt_lookup where (start<='";
        char decStr[25];
        sprintf(decStr, "%lu", dec);
//        cout<<"DEcstr:"<<decStr;
        strcat(query_string, decStr);
       // strcat(query_string, dec);
        strcat(query_string, "' and end >='");
        strcat(query_string, decStr);
        strcat(query_string, "')");

        if (mysql_query(cnx_init, query_string)!= 0){
          cout<<"query failed"<<endl;
          exit(1);
     }
     else {
          result_set = mysql_store_result(cnx_init);
          if (result_set == NULL){
               cout<<"mysql_store_result bombed"<<endl;
               exit(1);
          }
          else{
                    cout<<temp<<":";
                    if(mysql_num_rows(result_set) == 0){
                        for(int i =0 ; i<256; i++){
                            cout<<"0:";
                        }
                        cout<<endl;
                    }
                    while((row =
mysql_fetch_row(result_set)) != NULL){
                        for (ctr = 0;
ctr<mysql_num_fields(result_set); ctr++){
                            char region[20];
                            strcpy(region, row[ctr]);
                            cout<<"region:"<<region<<endl;
                            int numneighbors =
getNeighbors(region);
                        }
                    }

          }
     }

     return 0;
    }
```

```
    //displays all elements of the result set one by one
    //Useful for error checking
    void show_result_set(MYSQL_RES *in_result_set){

            while((row = mysql_fetch_row(in_result_set)) !=
NULL){
          for(ctr=0;ctr<mysql_num_fields(in_result_set);
ctr++){
               if (ctr>0){
                    cout<<"\t";
                                printf("%s",
row[ctr]!=NULL?row[ctr]:"Null-val");
               }
               cout<<"\n";
          }
            }

            if (mysql_errno(cnx_init) != 0){
          cout<<"Fetch row Error"<<":";

cout<<mysql_errno(cnx_init)<<mysql_error(cnx_init)<<endl;
          exit(1);
            }

     mysql_free_result(in_result_set);
    }

    //takes in class c address and gives back the decimal
form for it
    long ip2Dec(int i){
        long precalc = ((128*256)+173)*256;
        unsigned long Dec = (precalc + i)*256;
        return Dec;
    }


    //converts decimal form to ip address and returns class
c address
    int dec2Ip(unsigned long addr){
        unsigned long addr1 = addr/256;
        int subnet = ((addr1)%256);
        return subnet;
    }


      //finds out the neighbors, and puts them in an array
and prints to screen
    int getNeighbors(char region[20]){
```

```
        int neighbors= 0;
        //form the query
        char numSubnetQuery[99] = "Select start, end from
vt_lookup where region like '%";
        strcat(numSubnetQuery, region);
        strcat(numSubnetQuery, "%'");
        //run the query
        if (mysql_query(cnx_init, numSubnetQuery)!= 0){
            cout<<"subnet query failed"<<endl;
            exit(1);
        }
        else {
            subnet_result_set =
mysql_store_result(cnx_init);
            if (result_set == NULL){
                cout<<"mysql_store_result bombed"<<endl;
                exit(1);
            }
            else{
                neighbors =
mysql_num_rows(subnet_result_set);
                   //cout<<"Number of
neighbors"<<neighbors<<endl;
                // now check if there is more than 1 subnet
in any of the neighbors
                int neighbor_weights[256][4];
                int temp_neighbor_weights[100];
                int partitionArray[100][3];//keep
partition-parent information
                int i=0;// used to maintian count of
neighbor_weights
                int j=0;// used to maintian count of
temp_neighbor_weights
                int partition = 0; // used to track the
router partition
                int count =0;

                while((row =
mysql_fetch_row(subnet_result_set)) != NULL){
                    char *pEnd;
                    unsigned long start = strtoul(row[0],
&pEnd, 0);
                    unsigned long end = strtoul(row[1],
&pEnd, 0);
                    if ((end-start) > 256){
                        //cout<<"More than one subnet
here:";
```

```
                        int numIntSubnets = (end-
start)/255;
                       // cout<<numIntSubnets<<"subnets:";
                        int firstsubnet = dec2Ip(start);
                       // cout<<firstsubnet;

addNeighborWeights(neighbor_weights, firstsubnet,
numIntSubnets, 1, partition, i++, partitionArray, count);
                        for (int n =0; n != (numIntSubnets
- 1);n++){

addNeighborWeights(neighbor_weights, ++firstsubnet,
numIntSubnets, 1, partition, i++, partitionArray, count);
                         }
                        //cout<<"\t";
                        partition++;
                    }
                    //handle case when only 1 subnet exists
                    else{
                        // make sure that it is not a
single host connected to the router. e.g. DNS host
                        if ((end-start) > 10 && (end-start)
< 256){
                            int subnet = dec2Ip(start);

                            int flag = 0; // to check for
duplicate values
                            for(int n=0; n<j;n++){
                                if(temp_neighbor_weights[n]
== subnet){
                                    flag = 1;
                                }
                            }
                            // only adds if not duplicate
                            if (flag == 0){
                                temp_neighbor_weights[j++]
= subnet;
                            }
                        }
                     }

                }
                //now transfer all the temporary subnets to
this
                for (int n = 0; n<j &&
temp_neighbor_weights[n] < 256 && temp_neighbor_weights[n]
>= 0; n++){
```

```
                    addNeighborWeights(neighbor_weights,
temp_neighbor_weights[n], j-1, 0, -1, i++, partitionArray,
count);
                }

                cout<<"Neighbor array:";
                int n = 0;
                while (n<i && neighbor_weights[n][0] !=
666){
                    cout<<"["<<neighbor_weights[n][0];
                    cout<<"|"<<neighbor_weights[n][1];
                    cout<<"|"<<neighbor_weights[n][2];
                    cout<<"|"<<neighbor_weights[n][3];
                    cout<<"]";
                    n++;
                }
                cout<<"partitionarray:";
                int temp = 0;
                while(temp <count &&
partitionArray[temp][0] != 666){
                    cout<<"["<<partitionArray[temp][0];
                    cout<<"|"<<partitionArray[temp][1];
                    cout<<"|"<<partitionArray[temp][2];
                    cout<<"]";
                    temp++;
                }
            }
        }
        cout<<endl;
        return neighbors;
    }



    //find out th epartition etc of the neighbor and add it
to the array
    void addNeighborWeights(int neighbor_weights[256][4],
int subnetnumber, int numIntSubnets, int level, int
partition, int row, int partitionArray[100][3], int
&count){
        if (subnetnumber>=0 && subnetnumber < 256){// don't
do any thing if subnet umbr is greater than 256
            int flag = 0; // to check for duplicates
            for(int n =0; n<row;n++){
                //if duplicate exists
                if(neighbor_weights[n][0] == subnetnumber){
```

```
                    if (level != 0){ // if duplicate exists
and level is 0, we don't want to do anything
                        //increase it's level
                        neighbor_weights[n][2] += 1;
                        // whichever partition has the
smaller number of subnets, the subnet belongs to it
                        if(neighbor_weights[n][1] >
numIntSubnets){
                            // now the subnet at higher
level has weakened. Therefore we need to reduce the number
of neighbor subnets there
                            for(int x =0; x < row; x++){
                                if ((neighbor_weights[x][3]
== neighbor_weights[n][3]) && (neighbor_weights[x][2] ==
(neighbor_weights[n][2]-1))){
                                    neighbor_weights[x][1]
-= 1;
                                }
                            }


add2PartitionArray(partitionArray, numIntSubnets,
neighbor_weights[n], partition, count);

                            neighbor_weights[n][1] =
numIntSubnets;
                            neighbor_weights[n][3] =
partition;

                        }
                    }

                    flag = 1;
                }
            }
            //duplicate does not exist
            if (flag == 0){
                neighbor_weights[row][0] = subnetnumber;
                neighbor_weights[row][1] = numIntSubnets;
                neighbor_weights[row][2] = level;
                neighbor_weights[row][3] = partition;
                neighbor_weights[row+1][0] = 666;
                add2PartitionArray(partitionArray,
numIntSubnets, neighbor_weights[row], partition, count);
            }
        }
    }
```

```
    //manipulates the partition array
    void add2PartitionArray(int partitionArray[100][3], int
numIntSubnets, int neighbor_weights[3], int partition, int
&count){
        int flag1 = 0;
        //now check if the artition exists in the partition
array
         for(int r=0; r<count && flag1 !=1;r++){
             if(partitionArray[r][0] == partition){
                 flag1 =1;
              }
          }
          //if partition does not exist, add to partition
array along with parent information
           if (flag1 != 1){
               partitionArray[count][0] = partition;
               partitionArray[count][1] = numIntSubnets;
               // if the added neighbor is at level 1, it's
parent is 0
               if (neighbor_weights[2] == 1){
                    partitionArray[count][2] = -1;//forced
partition number
               }
               // if added neighbor is at level 0
               else if (neighbor_weights[2] == 0){
                    partitionArray[count][2] = -5;//NOTE:-5
should not exist as a partition ever. This is just so that
we can distinguish those at level 0
               }
               //this means it's being pushed down the
level
               else{
                    partitionArray[count][2] =
neighbor_weights[3];//value of parent
               }
               partitionArray[count+1][0] = 666;
               count++;
            }
    }


    //calculation of actual weights
    //NOTE::This is the heart of the topological code and
in fact the reason for doing all this
    void calculateWeights(int neighbor_weights[256][4],
float weights[256], int partitionArray[100][3], int
myself[4]){
```

```
        for(int i = 0; i<256; i++){
            int flag = 0;
            for(int n=0; neighbor_weights[n][0] !=666 &&
flag != 1; n++){
                if (neighbor_weights[n][0] == i){
                    flag = 1;
                }
            }
            //found it
            if (flag == 1){
                //check if they are in the same partition
                if(myself[3] == neighbor_weights[n][3]){
                    weights[i] = 1/neighbor_weights[n][1];
                }
                //check if one is the child of other
                else if(searchpartition(myself[3],
neighbor_weights[n][3], partitionArray) == 0)
            }
            //not in neighborhood
            else{
                weights[i] = 0.0;
            }

        }
    }

    //will return 0 if one is the other's child
    int searchPartition(int firstsubnet, int secondsubnet,
int partitionArray[100][3]){
        int temppartitionarray1[3];
        int temppartitionarray2[3];
        int intermediatepartition[3];
        int returnvariable = 1;

        //find partition numbers for both subnets
        inpartition(firstsubnet, partitionArray,
temppartitionarray1);
        inpartition(secondsubnet, partitionArray,
temppartitionarray2);

        //first see if the first subnet exists in the
second subnets path to core router
        while (intermediatepartition[2] !=
temppartitionarray1[0]){
            inpartition(intermediatepartition[2],
partitionArray, intermediatepartition);
        }
        // if it exists will exit while loop prematurely
```

```
    }

    void inpartition(int subnet, int
partitionArray[100][3], int temppartitionarray[3]){
            int flag = 0;
            for(int n=0; partitionArray[n][0] !=-1 && (flag
!= 1 || flag1 !=1); n++){
                if (partitionArray[n][0] == subnet){
                    temppartitionarray = partitionArray[n];
                    flag = 1;
                }
            }

    }
};
```

Appendix A.4: checks.cpp

```
#include <iostream>
#include <string>

using namespace std;
#define MAX_INT 2147483647

class valueCheck{
     private:
          enum {NOT_INT, INT};
     public:
     valueCheck(){}
     ~valueCheck(){}
     bool is_str(string n)
     {
          if (n.empty()){
               return false;
          }
          else {
               return true;
          }
     }

     bool is_char(char n){
          if ((n >= 'A' && n<='Z') || (n>='a' && n <= 'z'))
          {
               return true;
          }
          else {
               return false;
          }
     }

     bool is_int(int n){
          if ((n > 0) && (n < MAX_INT)){
               return true;
          }
          else{
               return false;
          }
     }

        /**
        * Function fatalError
* input string
* output void(with exit status)
**/
```

```
void fatalError(string errString){
     valueCheck C;
     if (!C.is_str(errString)){
          cout<<"Error String passed contains invalid
parameters"<<endl;
          exit(1);
     }
     cout<<"An error occured in processing:"<<endl;
     cout<<errString<<endl;
     exit(1);
}


};
```

7.3 Appendix B

Appendix B.1: ip2vtgrid.php

```
<?
/*************************************************************
************************************
*    This code is meant for conversion of IP numbers and to
a specific grid cell on         *
*    Virginia Tech's grid layout map available at
http://www.unirel.vt.edu/map/map-overall.html*
*    This can be further used for display
purposes(functionality not provided).          *
*    This code connects to a database called blaster with
table and column specifications as   *
*    described in the mysql database file.
                *
*    In case of questions or comments, please feel free to
contact Rishi Pande at rpande@vt.edu*
*    This code is distributed within the confines of the
GPL and come with no guarantees,      *
*    implied or percieved.
            *
*************************************************************
************************************/

// function to provide database connectivity
function connect($db) {
        $DB_HOST="localhost";
        $DB_NAME=$db;
        $DB_USER="****";
        $DB_PASS="******";
//global $conn, $DB_HOST, $DB_USER, $DB_PASS, $DB_NAME;
        if (!$conn=mysql_connect($DB_HOST,$DB_USER,
$DB_PASS)){
                echo "Connect error:".mysql_error();
        }
        if(!mysql_select_db($DB_NAME,$conn)){
                echo "Select database error".mysql_error();
        }
     return $conn;
}

$conn_2 = connect('vt_data');
$select_ips = "select count(*) as count, source from witty
where source like ";
$select_ips .= "'128.173.%' group by source";
$ip_query = mysql_query($select_ips, $conn_2);
```

```
$conn = connect('blaster');
$count= array();
$comps = array();
echo "grid;count\n";
while ($ip_result_array = mysql_fetch_array($ip_query)){
     $ip = $ip_result_array['source'];
     //$time = $ip_result_array['time'];
     $part = explode('.', $ip);
     $part_dec = 0;
     foreach($part as $part){
          $part_dec = ($part_dec + $part)*256;
     }
     $decimal = $part_dec/256;
     $select_short_name = "Select building from vt_lookup
where start <=";
     $select_short_name .= $decimal." and end >=" .
$decimal;
     $short_name_query = mysql_query($select_short_name,
$conn) or die("Invalid Query:".mysql_error());
     $short_name_array =
mysql_fetch_array($short_name_query);
     $short_name = $short_name_array['building'];
     $select_long_name = "Select name from vt_region where
building like '%";
     $select_long_name .= $short_name."%'";
     $long_name_query = mysql_query($select_long_name,
$conn) or die("Invalid vt_region Query:".mysql_error());
     $long_name_array =
mysql_fetch_array($long_name_query);
     $long_name = $long_name_array['name'];
     $select_get_grid = "Select vt_row, vt_column from
building_grid where name like '%";
     $select_get_grid .= $long_name."%'";
     $get_grid_query = mysql_query($select_get_grid,
$conn);
     $get_grid_array = mysql_fetch_array($get_grid_query);
     $get_grid_row = $get_grid_array['vt_row'];
     $get_grid_col = $get_grid_array['vt_column'];
     $select_num_comps = "Select service_count, service_qty
from building_computers where name like
'%".$long_name."%'";
     $num_comps_query = mysql_query($select_num_comps,
$conn) or die("Invalid query:".mysql_errno());
     $tot_comps = 0;
     while($num_comps_row =
mysql_fetch_array($num_comps_query)){
          if($num_comps_row['service_type'] == "Ethernet"){
```

```
                $temp_comps = $num_comps_row['service_qty']
* 0.85;
          }
          else{
                $temp_comps = $num_comps_row['service_qty'];
          }

          $tot_comps += $temp_comps;
     }
     $comps[$get_grid_row][$get_grid_col] = $tot_comps;

     if (!$count[$get_grid_row][$get_grid_col]){
          $count[$get_grid_row][$get_grid_col] = 1;
     }
     else{
          $count[$get_grid_row][$get_grid_col] += 1;
     }
}
for($i = 'A'; $i <= 'R'; $i++){
     for($j =1;$j <= 18; $j++){
          echo $i.$j.";";

          if($count[$i][$j]){
                echo $count[$i][$j]/$comps[$i][$j];
          }
          else{
                echo "0";
          }
          echo "\n";
     }
}

//echo "</html>"
?>
```

Appendix B.2: ip2vttopo.php

```
<?
/*************************************************************
*************************************
*    This code is meant for conversion of IP numbers and to
a specific subnet on            *
*    Virginia Tech's network layout map *
*    This can be further used for display
purposes(functionality not provided).          *
*    This code connects to a database called blaster with
table and column specifications as   *
*    described in the mysql database file.
                *
*    In case of questions or comments, please feel free to
contact Rishi Pande at rpande@vt.edu*
*    This code is distributed within the confines of the
GPL and come with no guarantees,      *
*    implied or percieved.
           *
*************************************************************
*************************************/

// function to provide database connectivity
function connect($db) {
        $DB_HOST="localhost";
        $DB_NAME=$db;
        $DB_USER="****";
        $DB_PASS="******";
//global $conn, $DB_HOST, $DB_USER, $DB_PASS, $DB_NAME;
        if (!$conn=mysql_connect($DB_HOST,$DB_USER,
$DB_PASS)){
                echo "Connect erroe:".mysql_error();
        }
        if(!mysql_select_db($DB_NAME,$conn)){
                echo "Select database error".mysql_error();
        }
     return $conn;
}

$conn_2 = connect('vt_data');
$select_ips = "select count(*) as count, source, time from
slammer where source like ";
$select_ips .= "'128.173.%' group by source having count >
1 order by time";
$ip_query = mysql_query($select_ips, $conn_2);
$conn = connect('blaster');
$count= array();
```

```
$comps = array();
echo "subnet;count\n";
while ($ip_result_array = mysql_fetch_array($ip_query)){
     $ip = $ip_result_array['source'];
     $time = $ip_result_array['time'];
     $part = explode('.', $ip);
     $part_dec = 0;
     $tip = 1;
     foreach($part as $part){
          if($tip == 3){
               $subnet = $part;
          }
          $part_dec = ($part_dec + $part)*256;
          $tip++;
     }
     $decimal = $part_dec/256;
     $select_short_name = "Select building, start, end from
vt_lookup where start <=";
     $select_short_name .= $decimal." and end >=" .
$decimal;
     $short_name_query = mysql_query($select_short_name,
$conn) or die("Invalid Query:".mysql_error());
     $short_name_array =
mysql_fetch_array($short_name_query);
     $short_name = $short_name_array['building'];
     $start = $short_name_array['start'];
     $end = $short_name_array['end'];
     $select_long_name = "Select name from vt_region where
building like '%";
     $select_long_name .= $short_name."%'";
     $long_name_query = mysql_query($select_long_name,
$conn) or die("Invalid vt_region Query:".mysql_error());
     $long_name = mysql_result($long_name_query, 0);
     $select_num_comps = "Select service_count, service_qty
from building_computers where name like
'%".$long_name."%'";
     $num_comps_query = mysql_query($select_num_comps,
$conn) or die("Invalid query:".mysql_errno());
     $tot_comps = 0;
     while($num_comps_row =
mysql_fetch_array($num_comps_query)){
          if($num_comps_row['service_type'] == "Ethernet"){
               $temp_comps = $num_comps_row['service_qty']
* 0.85;
          }
          else{
               $temp_comps = $num_comps_row['service_qty'];
          }
```

```
          $tot_comps += $temp_comps;
     }
     if(floor(($end-$start)/256) != 0){
          $comps[$subnet] = $tot_comps / floor(($end-
$start)/256);
     }
     else{
          $comps[$subnet] = $tot_comps;
     }
     if (!$count[$subnet]){
          $count[$subnet] = 1;
     }
     else{
          $count[$subnet] += 1;
     }
}
for($i = 0; $i <= 255; $i++){
          echo $i.";";
          if($i<100){
          $i = "0".$i;
          }

          if($count[$i]){
               if($comps[$i] == 0){
                    echo $count[$i]/100;
               }
               else{
                    echo $count[$i]/$comps[$i];
               }
          }
          else{
               echo "0";
          }
          echo "\n";
}

//echo "</html>"
?>
```

Appendix C

Appendix C.1: geostat.sas

```
options nodate pageno=1 ls=80;


data geoplot;
infile 'U:\TAConsulting\Rishi\New Folder\geoplot.csv'
dlm=',' firstobs=2;
input row col count_blaster count_slammer count_slapper;
run;

proc print;
run;
data geonew;
set geoplot;
where count_blaster<>-1;
countB=count_blaster*100;
countM=count_slammer*100;
countP=count_slapper*100;
indB=(countB>0);
indM=(countM>0);
indP=(countP>0);
ind=indB+2*indM+4*indP;
run;


proc print;
var row col count_blaster count_slammer count_slapper indB
indM indP ind;
run;
proc gplot data=geonew;
plot row*col=ind/ frame cframe=ligr haxis=axis1
vaxis=axis2;
symbol1 v=dot color=blue i=none;
symbol2 v=dot color=red i=none;
symbol3 v=dot color=green i=none;
symbol4 v=dot color=purple i=none;
symbol5 v=dot color=yellow i=none;
axis1 minor=none;
axis2 minor=none label=(angle=90 rotate=0);
run;

quit;
```

Appendix C. 2: topostat.sas

```
*bringing topology and virus incidents together - checking
compatibility;
%macro Moran(virus);
proc sort data=&virus;
by k;
proc means data=&virus noprint;
var fraction;
output out=fracmean mean=fracmean;
run;
data &virus;
set &virus;
if _n_=1 then set fracmean;
fraccenter=fraction-fracmean;
fraccenter2=fraccenter**2;
run;
data Moran&virus;
merge topology &virus;
by k;
run;
proc print data=Moran&virus;
where j=1 & fraction=. or w=.;
run;
%mend Moran;
%Moran(blaster);
%Moran(slammer);
%Moran(slapper);


*Further analysis;
%macro I(virus);
data vert&virus;
set &virus (rename=(fraccenter=fraccenter_o));
j+1;
keep j fraccenter_o;
run;
proc sort data=Moran&virus;
by j;
data Moran&virus;
merge Moran&virus vert&virus;
by j;
run;
data Moran&virus;
set Moran&virus;
prod=w*fraccenter*fraccenter_o;
run;
proc means data=Moran&virus sum;
```

```
var w prod;
output out=sum sum=w prod;
run;
proc means data=&virus;
var fraccenter2;
output out=frac2 mean=fraccenter2;
run;
data a;
merge sum frac2;
I = prod/w/fraccenter2;
sigma2I=fraccenter2;
z=(I+1/(_freq_-1))/sqrt(sigma2I);
probz=1-probnorm(z);
run;
proc print data=a;
run;
   %mend vert ;
```

Appendix D

Appendix D.1: Permission to reproduce Figure 2

From: cshannon@caida.org
Subject: **Re: permission to reproduce image in my thesis**
Date: April 27, 2004 12:55:59 PM EDT
To: rpande@vt.edu
Cc: dmoore@caida.org
Return-Path: <cshannon@login.caida.org>
Received: from steiner.cc.vt.edu (evil-steiner.cc.vt.edu [10.1.1.14]) by lyta.cc.vt.edu (iPlanet Messaging Server 5.2 Patch 1 (built Aug 19 2002)) with ESMTP id <0HWU00LD79PG4K@lyta.cc.vt.edu> for rpande@ims-ms-daemon (ORCPT rpande@vt.edu); Tue, 27 Apr 2004 12:56:05 -0400 (EDT)
Received: from login.caida.org (login.caida.org [192.172.226.78]) by steiner.cc.vt.edu (MOS 3.4.6-GR) with ESMTP id AQR26058; Tue, 27 Apr 2004 12:56:01 -0400 (EDT)
Received: from login.caida.org (localhost [127.0.0.1]) by login.caida.org (8.12.10/8.12.10) with ESMTP id i3RGtxbV062670 (version=TLSv1/SSLv3 cipher=EDH-RSA-DES-CBC3-SHA bits=168 verify=NO); Tue, 27 Apr 2004 09:55:59 -0700 (PDT)
Received: (from cshannon@localhost) by login.caida.org (8.12.10/8.12.10/Submit) id i3RGtx27062669; Tue, 27 Apr 2004 09:55:59 -0700 (PDT)
In-Reply-To: <6C54D138-9869-11D8-92D0-000393793B04@vt.edu>
Message-Id: <20040427165559.GO59283@login.caida.org>
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Disposition: inline
User-Agent: Mutt/1.4.2.1i
X-Junkmail-Status: score=0/50, host=steiner.cc.vt.edu
References: 6C54D138-9869-11D8-92D0-000393793B04@vt.edu

Hi Rishikesh,

It's fine for you to use our image in your thesis. We'd also be very interested in the content of your thesis whenever it becomes available.

-C

--
Colleen Shannon
CAIDA/SDSC/UCSD - cshannon@caida.org

On Tue, Apr 27, 2004 at 12:39:16PM -0400, Rishikesh Pande wrote:

Hello,

I am graduate student at Virginia Tech working on my masters research "Using plant epidemiological methods to track network worms". I am currently writing my thesis draft and would like permission to reproduce a figure from your analysis of the Code-Red V 2 worm on the CAIDA network. The said figure is labeled figure 2 in the paper and shows the time-series spread of the CR v2 worm. I have attached a copy of the figure with this e-mail and it is also available on the Internet at
http://www.caida.org/analysis/security/code-red/gifs/cumulative-ts.gif

I plan on using the following reference for the figure.

Moore, Shannon (2001) "The spread of the code-red worm (CR v2)", CAIDA, <http://www.caida.org/analysis/security/code-red/coderedv2_analysis.xml>

My thesis will be available for reading from the Virginia Tech ETD (www.etd.vt.edu) library after graduation.

Thank you very much. I look forward to hearing back from you.

Sincerely,
Rishikesh Pande

Appendix D.2: Permission to reference Travis et al. (2003)

From: greg@iu.edu
Subject: **Re: REN-ISAC virus data + Slammer Analysis**
Date: May 3, 2004 9:52:13 AM EDT
To: rpande@vt.edu
Return-Path: <greg@iu.edu>
Received: from steiner.cc.vt.edu (evil-steiner.cc.vt.edu [10.1.1.14]) by lyta.cc.vt.edu (iPlanet Messaging Server 5.2 Patch 1 (built Aug 19 2002)) with ESMTP id <0HX500I1V574P0@lyta.cc.vt.edu> for rpande@ims-ms-daemon (ORCPT rpande@vt.edu); Mon, 03 May 2004 09:52:17 -0400 (EDT)
Received: from paintbird.ucs.indiana.edu (paintbird.ucs.indiana.edu [156.56.103.100]) by steiner.cc.vt.edu (MOS 3.4.6-GR) with ESMTP id ART37190; Mon, 03 May 2004 09:52:13 -0400 (EDT)
Received: from [156.56.103.3] (turtleneck.ucs.indiana.edu [156.56.103.3]) (authenticated bits=0) by paintbird.ucs.indiana.edu (8.12.10/8.12.10) with ESMTP id i43DpnuO025533 (version=TLSv1/SSLv3 cipher=RC4-SHA bits=128 verify=NO) for <rpande@vt.edu>; Mon, 03 May 2004 08:52:13 -0500
In-Reply-To: <4F6201D0-9C56-11D8-92D0-000393793B04@vt.edu>
Message-Id: <149B22EA-9D09-11D8-9186-000A95A7EA10@iu.edu>
Mime-Version: 1.0 (Apple Message framework v613)
X-Mailer: Apple Mail (2.613)
Content-Type: text/plain; format=flowed; charset=WINDOWS-1252
Content-Transfer-Encoding: quoted-printable
X-Junkmail-Status: score=0/50, host=steiner.cc.vt.edu
References: <1F17769B-423C-11D8-986B-000393BB2B90@iu.edu> 4F6201D0-9C56-11D8-92D0-000393793B04@vt.edu

Rishi,

That would be fine.

Thanks,

greg

On May 2, 2004, at 11:32 AM, Rishikesh Pande wrote:

> Dear Mr. Travis,
>
> You may remember that we spoke earlier in the year about analysis of worms. At the time, you passed me a document containing a SQL slammer report as seen by Abilene. I was wondering if I can have your permission to reference the report in my thesis. I plan to use the following citation:
>
> 1. Travis G., Balas, E., Ripley, D. and Wallace, S. (2003),"Analysis of SQL Slammer worm and it's effects on Indiana University and related institutions", University of Indiana, Technical Report
>
> Please let me know. Thank you very much.
>
> Sincerely,
> Rishi Pande

Curriculum Vitae

Rishikesh Pande

6000, Heather Drive, #K
Blacksburg, VA 24060

Phone No: 540-239-2948
Email: rpande@vt.edu

Education:

Master of Science (Computer Science), (GPA: 3.26) — May 2004 (Expected)
Virginia Polytechnic Institute and State University (Virginia Tech), Blacksburg, Virginia
Thesis: Advanced warning system prototype for agent based Intrusion Detection System

Bachelor of Engineering (Electronics) — May 2001
University of Mumbai, Mumbai, India

Computer Skills:

Languages
- C, C++, Java
- PERL, PHP, Unix shell scripting
- UML, ERD
- Visual Basic.NET, ASP.NET, ADO
- SQL, PL/SQL, XML

Applications
- Gdb, gcc, g++
- Rational Suite
- Visual Studio, Visual Studio.NET
- Zend, Project Builder
- LDAP, SSL, Tcpdump

Operating Systems
- All Windows Systems
- Mac OS 9, OS X
- Unix (Linux and BSD)

Course Projects:

Computer Supported Co-op Work	* Usability evaluation of BrightSuite (Intranet Team-collaboration software) * Design location based collaboration system for spatially oriented task co-ordination
Software Engineering	* Design a client-server J2EE architecture for a dot-com company using UML
Information Storage and Retrieval	* Implement a very fast vector-space search engine for the Open Archives Initiative
Operating Systems	* Simulation of Multi-programming batch operating system * Implementation of Multi-player Othello game
Digital Libraries	* Design and development of an archeological digital library (NSF funded project)

Coursework:

- Decision Support Systems: design and evaluation
- Usability Engineering
- Information Visualization
- System and Network Security (Audit)

Work Experience:

Technical Assistant, Digital Library and Archives, Virginia Tech, Blacksburg, VA (continued) — Aug 2003- present, Sept 2002 – May 2003
- Electronic Thesis and Dissertation (ETD-db) ver 2: enhance product with XML, SSL, OOP, etc
- E-docs: design, development & testing application for university research papers archive.
- ETD- design and development for the electronic theses submission web-pages
- Misc: system administration tasks, maintain & update legacy databases and websites

Research Student, Information Technology Security Lab, Virginia Tech, Blacksburg, VA — Aug 2003-present
- Project: Advanced warning system prototype for D-Shield Intrusion Detection System
- Research and design early attack warning system for Distributed Intrusion Detection System
- Development of LAN-geographic prediction agent based on epidemiological models

Intern, University Libraries, Virginia Tech, Blacksburg, VA — June 2003- Aug 2003
- Security analysis/ management for intranet using penetration tests, setup of firewall (ipfw), IDS
- International Archive of Women in Architecture -design and modify database, implement new design for website user/admin interfaces, design and implement weight based search engine
- Develop in-house applications for auto-updates of OS x boxes, bulk updates to databases

Research Student, Center for Human-Computer Interaction, Virginia Tech, Blacksburg, VA — Jan 2002- May 2003
- Project: Location Based Services for MOOsburg
- Use scenario- based design methodologies to implement a wireless location enabled reminder

Intern, **Motech Software Pvt. Ltd., Mumbai, India** — Aug 2000- April 2001
- Involved in the design, development & testing of a Convergence Billing System-MDR
- Developed prototype of real-time billing system for convergent Internet Service Providers

Presentations/ Activities:

- Torgersen Research Excellence award (M.S. -Poster), Virginia Tech — 2004
- Presentation: “MIDS: Taking DIDS into the enterprise”, SANSFIRE, Monterey, CA — 2004
- Director, Travel Fund Program, Graduate Students Assembly, Virginia Tech — 2002-2003