

Air Force Institute of Technology

**AFIT Scholar**

---

Theses and Dissertations

Student Graduate Works

---

4-2006

## **An Interactive Relaxation Approach for Anomaly Detection and Preventive Measures in Computer Networks**

Garrick A. Bell

Follow this and additional works at: <https://scholar.afit.edu/etd>



Part of the [Applied Mathematics Commons](#), and the [Information Security Commons](#)

---

### **Recommended Citation**

Bell, Garrick A., "An Interactive Relaxation Approach for Anomaly Detection and Preventive Measures in Computer Networks" (2006). *Theses and Dissertations*. 3285.

<https://scholar.afit.edu/etd/3285>

This Thesis is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact [AFIT.ENWL.Repository@us.af.mil](mailto:AFIT.ENWL.Repository@us.af.mil).



**AN ITERATIVE RELAXATION APPROACH TO ANOMALY DETECTION  
AND PREVENTIVE MEASURES IN COMPUTER NETWORKS**

THESIS

Garrick A. Bell

AFIT/GCE/ENG/06-01

**DEPARTMENT OF THE AIR FORCE  
AIR UNIVERSITY**

***AIR FORCE INSTITUTE OF TECHNOLOGY***

---

---

**Wright-Patterson Air Force Base, Ohio**

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

AFIT/GCE/ENG/06-01

**AN ITERATIVE RELAXATION APPROACH FOR ANOMALY DETECTION  
AND PREVENTIVE MEASURES IN COMPUTER NETWORKS**

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the  
Degree of Master of Science in Computer Engineering

Garrick A. Bell, BS

June 2006

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AFIT/GCE/ENG/06-01

**AN ITERATIVE RELAXATION APPROACH FOR ANOMALY DETECTION  
AND PREVENTIVE MEASURES IN COMPUTER NETWORKS**

Garrick A. Bell, BS

Approved:

/signed/  
\_\_\_\_\_  
Dr. Guna Seetharaman (Chairman)

4/28/06  
Date

/signed/  
\_\_\_\_\_  
Dr. Richard Raines (Member)

4/28/06  
Date

/signed/  
\_\_\_\_\_  
Dr. Rusty Baldwin (Member)

4/28/06  
Date

### **Abstract**

It is proposed to develop a framework of detecting and analyzing small and widespread changes in specific dynamic characteristics of several nodes. The characteristics are locally measured at each node in a large network of computers and analyzed using a computational paradigm known as the Relaxation technique. The goal is to be able to detect the onset of a worm or virus as it originates, spreads-out, attacks and disables the entire network. Currently, selective disabling of one or more features across an entire subnet, e.g. firewalls, provides limited security and keeps us from designing high performance net-centric systems. The most desirable response is to surgically disable one or more nodes, or to isolate one or more subnets.

The proposed research seeks to model virus/worm propagation as a spatio-temporal process. Such models have been successfully applied in heat-flow and evidence or gestalt driven perception of images among others. In particular, we develop an iterative technique driven by the self-assessed dynamic status of each node in a network. The status of each node will be updated incrementally in concurrence with its connected neighbors to enable timely identification of compromised nodes and subnets. Several key insights used in image analysis of line-diagrams, through an iterative and relaxation-driven node labeling method, are explored to help develop this new framework.

## **Acknowledgments**

In loving memory of my mother .

R.I.P October 1946 – January 2006

You will always be remembered, but deeply missed.

Garrick A. Bell

## Table of Contents

	Page
Abstract.....	iv
Acknowledgments.....	v
List of Figures.....	ix
List of Tables .....	xi
I. Introduction .....	1
1.1 Background.....	1
1.2 Problem Statement.....	8
1.3 Research Objectives/Questions/Hypotheses .....	8
1.4 Research Focus.....	11
1.5 Investigative Questions .....	12
1.6 Approach .....	14
1.7 Assumptions/Limitations.....	14
1.8 Implications .....	15
1.9 Thesis Overview .....	16
II. Literature Review .....	17
2.1 Chapter Overview.....	17
2.2 Description .....	17
2.3 Computer Worms .....	18
2.3.1 <i>Basic Components of a Worm</i> .....	19
2.3.2 <i>Scanning and Attack Patterns</i> .....	21
2.3.3 <i>Payload Methods</i> .....	25
2.3.4 <i>Future of Worms</i> .....	26



2.4 Worm Simulation .....	28
2.4.1 Blaster Worm Simulation.....	30
2.4.2 MyDoom Worm Simulation.....	31
2.4.3 Slammer Worm Simulation .....	34
2.4.4 Sasser Worm Simulation.....	35
2.5 Honeypots.....	38
2.5.1 Advantages and Disadvantages .....	38
2.5.2 Types of Honeypots.....	39
2.5.3 Value of Honeypots.....	41
2.6 Summary.....	42
III. Architectural Elements of Attack Resilient Networks.....	44
3.1 Chapter Overview.....	44
3.2 Hardware-based Prevention and Detection .....	45
3.3 Relevant Research .....	46
3.4 Potential Indicators of Malicious Activities .....	49
3.5 System Behavioral Description .....	50
3.6 Summary.....	51
IV. Label Propagation and Network Labeling.....	53
4.1 Chapter Overview.....	53
4.2 Relaxation Synopsis .....	54
4.2.1 Image Analysis.....	54
4.2.2 Network Analysis .....	57
4.3 Node Labeling and Propagation .....	59

4.4 Canonical Node Types .....	67
4.5 Infection Propagation Scenario .....	73
4.6 Future Modeling and Labeling Issues .....	77
4.7 Summary.....	78
V. Future Research and Recommendations .....	80
5.1 Chapter Overview.....	80
5.2 System Behavior.....	80
5.3 Auxiliary Support Network .....	87
5.4 System Policies.....	88
5.5 System Components .....	92
5.6 Summary.....	95
VI. Conclusions.....	97
6.1 Chapter Overview.....	97
6.2 Conclusions of Research .....	97
6.3 Significance of Research .....	98
6.4 Summary.....	98
Appendix 1: Full Flow Chart of ADPU System .....	99
Bibliography .....	100

## List of Figures

	Page
Figure 1: Necker Cube – Is the dot at the rear bottom left or the front bottom left? .....	12
Figure 2: Screenshot of Symantec Worm Simulator .....	29
Figure 3: Blaster Worm Cleaned, Patched, and Infected Rates .....	31
Figure 4: MyDoom Cleaned, Patched, Dormant, and Infected Rates .....	33
Figure 5: Slammer Worm Cleaned, Patched, and Infection rates .....	35
Figure 6: Sasser Worm Clean, Patched, and Infection rates .....	37
Figure 7: Edge Relaxation Algorithm [2] .....	55
Figure 8: Edge Notation [2] .....	55
Figure 9: Edge relaxation example images [2] .....	57
Figure 10: Line Labels and Junction labels for trihedral polyhedron .....	60
Figure 11: Consistent Labeling Example .....	62
Figure 12: Two possible labels for the Y junctions of the cube [6] .....	63
Figure 13: Depending on the order of junction analysis, a complicated or simple tree is created to search for a valid solution. ....	64
Figure 14: Network layout and user interaction for simulated scenarios .....	69
Figure 15: Example network scenario for labeling and relaxation. ....	73
Figure 16: Weighted connections, compatibility graph, and virus spreading events for the network example of relaxation methods .....	75
Figure 17: Typical network block diagram .....	82
Figure 18: FPGA implemented network anomaly detection block diagram .....	83

Figure 19: Single Layer Small-World Network model with $L=24$ , $k=3$ , and $S=4$ [12] .....	84
Figure 20: Two-layer Small-World Network model [12] .....	85
Figure 21: Anomaly Detection and Prevention Unit implemented in a small network .....	86
Figure 22: Scenario 1 - Incoming Anomalous Activity .....	89
Figure 23: Scenario 2 - Outbound Anomalous Activity .....	90

## List of Tables

	Page
Table 1: FPGA vs. software regular expression performance on 1MB data set sent in 1kB chunks [13] .....	47
Table 2: FPGA vs. software regular expression performance on 16MB data set sent in 16kB chunks [13].....	47
Table 3: Edge Type Rules.....	56
Table 4: All 18 possible junctions for trihedral polyhedron .....	61
Table 5: Two-node virus propagation from e-mail and file execution .....	70
Table 6: Propagation paths.....	72
Table 7: Compatibilities between users - Larger value represents most used interaction method.....	74

# **An Iterative Relaxation Approach for Anomaly Detection and Preventive Measures in Computer Networks**

## **I. Introduction**

### **1.1 Background**

Relaxation is an iterative computational paradigm used in many spatio-temporal problems such as: the study of heat-flow in a bounded space, evidence based incremental perception of images, resource allocation, and constraint propagation in inference networks. This thesis describes the development of a framework to model the propagation of virus and worms in a computer network using the Relaxation techniques. We assume that the network is comprised of several functionally heterogeneous nodes with varied degrees of mutual trust relationship amongst them. Simple simulations have been used to illustrate the point; however, large-scale experimental verification is not included in the scope of the present study.

The propagation tactics of viruses and worms in a network closely resemble the sequential processes inherent to heat-flow in bounded space. These processes can be characterized by a set of isolated heat sources and a well defined set of immutable spatial location also known as boundary conditions. The objective is to compute the steady state distribution of temperature in the volume and the transient behavior at each location after a heat source has been introduced into the volume.

Several fortified nodes in a network define the boundary conditions; the un-secured ones constitute a medium of propagation, and each of the infected nodes play the role of a heat source. The propagation itself is modeled as a diffusion or conduction

process whose dynamics are characterized by pair-wise trust and traffic between any two connected nodes in the network. Thus, the scope of studies in virus propagation bears a similarity to the modeling and analysis of heat-flow. The relaxation paradigm, once again, has been successfully applied to heat-flow computations, principally due to the inherent potential for parallel computation made feasible by this approach.

The principle concept that govern the physics of heat flow in a bounded volume, free from active sources or sinks, states that the temperature will reach a steady state such that the spatio-temporal changes will eventually vanish (or become minimal), except at locations where there is a source or a sink. Thus given a synaptic snapshot of the temperature across the field, one could predict the temperature in the next time frame, based only on the spatial derivatives of the local temperature and the effect of any local sources/sinks, as follows:

$$\frac{dT}{dt} = \left( \frac{\partial^2}{\partial^2 x} + \frac{\partial^2}{\partial^2 y} \right) T(x, y; t) + s(x, y; t).$$

The term  $s(x, y; t)$  represents the heat source. It is to be emphasized that the spatial derivatives are computed as a highly localized operation. Further reduction of this equation lends it self to the form:

$$T[x, y; t + (k + 1)dt] = T[x, y; t + k \cdot dt] + \varepsilon \cdot R[x, y; k \cdot dt],$$

where, the term  $R[x, y; k \cdot dt]$  is known as the residual error at location (x,y), whose value is exactly the same as the right hand side of the previous expression. A non-zero value indicates that the system has not reached local equilibrium. The scalar multiplier  $\varepsilon$  must be chosen carefully to allow the constraints to propagate across the grid; higher

values would promote rapid convergence based on local constraints, and smaller values would permit  $T(x,y;t+1)$  to be influenced by constraints originating from distant points. Note  $dt$  is very small compared to one.

The computation of self-assessed health indicators of each node is not as simple as the heat flow process introduced above. The difficulties originate from the fact such indicators are discrete in value and the spatial derivatives must be suitably modeled over the discrete space of nodes, i.e. node-indices. The discrete values of a scalar or vector measurement at each node is generally constrained by the corresponding measurements at its immediately connected neighboring nodes, in a manner that is governed by the nature of relationship between itself and each of its neighbor. Then, given the estimate of the state of affairs at each neighboring node and the node in question, one could estimate the extent to which the estimates agree or disagree. A quantitative assessment of such disagreement (residual incompatibility) is to be reckoned by adjusting the local estimate (at the present node). Such a procedure must be performed at all nodes iteratively in locked-steps until the residual conflicts vanish at all nodes. That is, the goal is to obtain local concurrence everywhere to produce a globally stable field of locally consistent assessments.

The above generalization is illustrated with a concrete example known as a line labeling algorithm for image understanding in the world of trihedral blocks. In the example provided later in the work, one can observe how label propagation can be aided with the use of a compatibility matrix at each node to determine a node's state from its neighbors. This matrix provides the information of how connections are made, and



which users communicate within the network. Our description of the line labeling problem and relaxation technique will be limited to the extent necessary to help explore and explain the basic elements of studying the dynamic state of nodes in a computer network, as a virus or worm propagates. The section on literature survey enlists a number of papers where the relaxation technique has been applied successfully. Our objective has been to provide the reader with parallels between the two problems and to model/analyze network security in a way that has not been realized before.

We draw parallels from a classic problem in computer vision known as discrete labeling of line drawings in a trihedral blocks world. In simple terms, the line labeling process assigns one of four possible interpretations to each line segment in an image, subject to the constraints that arise in corners where at most three edges concur. While the method is easily explained as a sequential process, the underlying evidence-driven perceptual grouping is highly conducive for distributed and parallel computation. We seek to emphasize that the local measurement made everywhere across the network are in fact mutually constrained by global context, refer to Chapter 4 for more details on these methods. In addition, a rich set of relationships, which if understood fully, can be exploited to develop a globally consistent inference about the network based only on dense localized measurements.

We propose to categorize the nodes in a network into one of several possible personalities. In a network of limited scope, such as a small office, these nodes are envisioned to be one of: work-station user privilege, work-station with super user privilege, a mail-server, a ftp-server, a print-server, simple-router, a gate-way, a ntp-

server, etc. Often, one physical computer may serve more than one of the above personalities. In that case, we would trivially model such nodes as virtual nodes. Initially we assume that all nodes are assigned a label indicating an uninfected state. The simplest possible way of labeling the health of a node is envisioned to include: uninfected-and-secured, uninfected-and-vulnerable, infected-contained, infected-contagious, and dysfunctional.

Preliminary analysis indicates that most common network installations are made of nodes whose personalities can be mapped to a finite set of functionalities / services offered by the nodes. At least in principle, it is possible to enumerate the relationship between any two nodes in a network, and measure the occurrence a definite operation on a node and its potential impact on the node's health as well as its connected neighbors. We must be more innovative in defining "connectedness" or "neighbors" to suit the context. For example, two nodes communicating through sftp may cause a tunnel between two subnets potentially permitting infection unnoticed by the firewalls!

Let  $\ell_1, \ell_2, \ell_3, \dots, \ell_L$  be  $L$  distinct mutually exclusive labels that can be assigned to a node characterizing its current state of health. One of these states, we will take into account an undecidable state as well. Let  $p(\ell | i)$  be the probability that the state  $\ell$  is assumed by a node  $i$ . Then given two nodes  $i$  and  $j$ , we could write the compatibility between those values as a set of mutual constraints in the form:

$$\Delta^k(\ell | i) = \sum_{\lambda} \sum_{j \in N_i} C(i, j, \ell, \lambda, d(i), d(j)) \cdot p^k(\lambda | j); \quad \ell, \lambda \in L.$$

Where  $\Delta(\ell | i)$  is the change in confidence at node  $i$  in state  $\ell$  and  $C$  is the compatibility between  $i$  and  $j$  their respective states  $\ell$  and  $\lambda$ .  $N_i$  is the neighborhood of  $i$ . From the equation,  $C(i, j, \ell, \lambda, d(i), d(j))$  can be further factored into the form  $R_H(i, j) \circ R_I(\ell, \lambda, i, j, d(i), d(j))$ . In the expression,  $R_H$  captures the effect of a hierarchical relationship between nodes.  $R_I$  models the relationship between infection symptoms at each node taking into account their dynamics,  $d(i)$  and  $d(j)$ . One would expect if an administrator of a network is infected, the infection would easily pass down to any user under the administrator's control. In the reverse sense though, any user would not be able to easily infect the administrator because of the hierarchical relationship between administrator and user, thus  $R_H(i, j) \neq R_H(j, i)$ . To predict the next most feasible state of node  $i$ , the normal of the labels can be divided by the current confidence, plus the change in label confidence. This yields the equation below, for updating the label of the node:

$$P^{(k+1)}(\ell | i) = \frac{P^k(\ell | i)[1 + \Delta^k(\ell | i)]}{\sum_{\ell} P^k(\ell | i)}$$

Suppose a workstation with super-user login has been infected. Then, the underlying cause (a worm) has maximum chance of infecting the other stations. In contrast, if the infected workstation is that of a less privileged user (perhaps due to a downloaded program, or file import) it does not have the same impact on the machines the user might subsequently access. This is particularly true when local network traffic is governed by a star-network with a firewall at the hub. The problem becomes serious if

the worm infected file has been introduced first to a file server, before it was opened for execution. In that case, some privileged workstation on the network may, by-design, be able to execute (accidentally by a super-user) the infected file. Thus, an imported file resident on file server is more lethal than an imported file resident on a user-workstation. Stated otherwise, the number of files created on a node over certain duration on a node, may have an impact on the health of its neighbors in the immediate future. Conversely, the existence of a firewall/virus scan on the hub annihilates the spreading. Several ideas similar to this will be enumerated individually to establish the baseline of compatibility of self-assessed health of nodes, taking into account the interaction between the TCP/UDP ports open on these nodes, and the services that are performed by each node.

Predicting where an infection will spread to is the next step in preventing a virus from encompassing a network. Virus propagation can be thought of as a heat flow problem. A source node is initially infected and neighboring nodes soon become infected as well. These “neighboring” nodes do not need to be physical neighbors, but neighbors in the sense that communication occurs between them on a normal basis. Just as a heat source in a room will gradually heat the entire room up to a maximum heat capacity, limited by boundary walls and their temperatures, a virus will spread through a network until it reaches set boundaries and conditions. An automated virus, known as a worm is the quickest way an attack can spread through a network.

Worms pose a larger threat than a single attacker does, mainly because the rate a worm spreads through a network is much faster than any person can gain access or control of the same number of computers. Since worms are automated, worm code

determines what communication ports will open, how the worm's payload is delivered, and once a host is infected, how to find more hosts. All of this can be done at rapid rate until the entire network is compromised. Furthermore, worms are persistent and therefore a network's defenses must be almost flawless, with all computers on that network having strong defenses as well. One weak link in a network and a worm can break in and still compromise each node on the network since most computers have a "trust" policy with other systems in the same network. Finally, although an "intelligent" worm has not yet been created, it is only a matter of time before one that can learn and adapt to its environment is created, instead of using set paths and backdoors to enter a host.

## **1.2 Problem Statement**

The problem we seek to attack is as follows: Identify a minimal set of parameters that can be measured locally at each node, to indicate the symptoms of a cyber attack. Identify a set of pair-wise relationships between nodes in a network that help characterize how attacks spread. Propose a computational strategy to perform a distributed computation with the locally measured symptoms, and help a global decision if a network is under attack. Finally, we look to address practical issues that will govern the acceptance and widespread usage of the resulting recommendations in the existing network.

## **1.3 Research Objectives/Questions/Hypotheses**

The objective of this research is to model virus propagation, similar to the heat flow diffusion method, using relaxation. This work will also build a foundation to exploit

the network traffic among various nodes to monitor the overall health of a network. Nearly every large-scale application involves some form of networked or distributed resources. Even a mundane task like scheduling a meeting has become automated through access to the remote calendars. Isolation through disconnecting from the network or closing all ports is not practical. Having open ports to a network is a must, but the vulnerability of computer ports needs to be monitored and secured. In future research, implementation of the model would use a field programmable gate array (FPGA)-based distributed firewalls (FPDF) to monitor and control network traffic actively in real-time. “Distributed” means that the proposed method is implemented in situ in each node in a network. The inevitability of using heterogeneous computer networks, and the issues associated with legacy systems, are well addressed by this approach.

For example, if a hardware firewall is the gatekeeper in a bank, the proposed mechanism is like a vault keeper. Each vault keeper knows exactly who the permitted users are, including authorized proxies. The system performs tasks beyond a typical firewall. It provides the basic block to implement a distributed system for network intrusion detection. In addition, it would insulate each machine from outsiders by concealing vulnerable information such as OS type, version, and patch-level, which are often the key for machine specific attacks. If an anomaly is detected, the FPDF can intervene and deny incoming or outgoing traffic. The device can also alert other nodes in the network about the compromise and update the health of the overall network. Often, the core operations at this stage are to compare a string to the most recently encountered

strings, as well as list of known virus elements. The list is dynamic. These are operations that have been efficiently implemented in FPGAs, which permit parallelism. These content matching servers, as they are called, perform well for normal network traffic. However, they fail against encrypted traffic, since the data signature matching cannot be performed.

The relaxation technique, applied to a computer network, assumes: locally consistent operations everywhere results in a globally consistent system [11]. Each node in a network determines its status by its own state and the states of its neighboring nodes and then takes appropriate security measures.

The states of a node are:

- 1) Fully secure with no vulnerabilities, since the system is not connected to the network
- 2) Secure, but with known and monitored vulnerabilities, such as open ports (Ideal state)
- 3) Thought to be fully secure, but contains vulnerabilities (worse case of false security, default initial configuration setting)
- 4) Not secure with known and unmonitored vulnerabilities
- 5) Not secure and open to everything purposely, such as a Honeypot

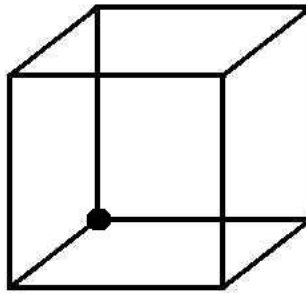
In a fully implemented system, all states will be used to describe a node's status in a network. If node #1 has known vulnerabilities (e.g., two open ports), while another system on the network has three vulnerabilities that are monitored and not monitored, system #1 has six possible ways of becoming infected or compromised in a "trust" based network. Without in-depth traffic monitoring, the status of node #1 should set to level 3 to reflect a possible attack through the node #2. With monitoring however, the security

policies can be changed to monitor for specific known vulnerabilities through node #2, while also checking traffic patterns for anomalies; possibly revealing other vulnerabilities. A more in depth review of the relaxation technique will be covered in Chapter 4.

#### **1.4 Research Focus**

This research focuses on establishing parallels between network virus propagation, heat flow and image labeling, topics that have been thoroughly studied and modeled in computer vision. The extent at which this topic would be effective in computer networks remains to be explored. In part, the reason for concern lies with the fact that a continuous function, such as heat-diffusion, does not readily import into the context of computer networks, which are largely characterized as a field of discrete events. Instead of viewing the problem as a discrete time, continuous function framework, it may be beneficial to model the problem as a discrete event or discrete-space framework. An example of such an instance is a Necker-cube, seen in Figure 1. One cannot discern from the image if the corner is supposed to be perceived as the rear bottom left or the front bottom left. By analyzing neighboring nodes and creating rules, a correct orientation of the cube can be observed.





**Figure 1: Necker Cube – Is the dot at the rear bottom left or the front bottom left?**

*Two stable interpretations can be perceived. Each interpretation is a stable labeling by the rules referred to in Figure 10, but the problem does not have a unique solution. If it is stated that only the inside of the bottom of the cube can be observed in the figure, then this evidence drives the perspective of the dot to be located in the rear bottom left corner. This concept of evidence-driven labeling can be related to computer networks and gathering information from nodes.*

### 1.5 Investigative Questions

Using a method proven from computer vision, but never applied to network security poses some questions. The method will improve virus detection, but to what extent? Also, after implementation, will the additional hardware in the network pose network performance and reliability problems that could affect the time it takes for the device to switch from one state to another? Although a hardware based system should be a great improvement over software detection systems, the added features the hardware can perform may offset these gains, but by how much?

One potential performance problem may result from placing the monitoring device in series between the node and its network. This device may cause enough delay that a user notices it. Scanning each packet in-depth could impose delays in the network traffic. Analysis of traffic throughput before and after the implementation of the hardware will measure the impact of this hardware.

A reliability problem will occur when the FPGA determines an attack is “successful,” or has passed the detection hardware and infected an end node. In the case of a false positive, the device would still reprogram itself to act as a reverse firewall, preventing any data from leaving the infected node. If an attack causes the device to become a reverse firewall, it effectively becomes a DoS (denial of service) attack. The user cannot use the node to reach any outside node. It is unknown at this time if the detection scheme allows DoS attacks, but the detection scheme should account for this potential problem.

A final problem is FPGA reprogramming time. If the number of gates in the FPGA prohibits the switching between two independent circuits, then the FPGA must be reprogrammed for each function. When the hardware detects an attack has breached the system, the FPGA reprograms itself to act as a reverse firewall. The problem arises during the time it must make this transition. During the transition, if the reprogramming takes too long, the worm/virus may have infected the system and propagated to another system in the network. When changing back to passive mode, the transition time is not as critical because the system must be disinfected and secured before the machine is allowed to connect with the network again.

While these questions will be discussed, simple simulations can only be used to illustrate the point. Until a large-scale experimental verification is performed, an exact measure of the performance gain cannot be produce. While the large-scale evaluation is beyond the scope of this work, the answers to the posed questions will be speculated later through the examples provided.

## **1.6 Approach**

Providing examples of image analysis and labeling techniques will help draw parallels between labeling images and labeling computer networks. Then the relaxation methods can be transferred to virus propagation with some modifications. Eventually though, to develop a device that protects against worms and viruses, while still allowing users access to services, several rules and methods need to be followed. An ideal device would employ countermeasures to automatically prevent the attack from continuing, but these countermeasures should not influence the user to a degree that renders the machine useless to them for extended periods. The analysis of connections to a node would reduce the negative impact of countermeasures on the system. Applying this technique with relaxation would allow users to slow the spread of worms and notify other nodes in the network of potential attacks. While this work does not cover how to slow or stop the spread of an attack, it does demonstrate the techniques used to predict where the virus / worm would spread to next, so the spread can be prevented by other means. The relaxation approach is used in this research to determine the health of the nodes in a network.

## **1.7 Assumptions/Limitations**

In this research, worms are assumed to be capable of evading current network security techniques by spreading rapidly with little or no detection. Creators of operating systems will develop systems with features that have flaws and exploits. While patches will be available when an exploit is found, some of these will not be applied in time, increasing the chance of vulnerability across the network.

The device proposed for future work is not operating system dependant. If it were, it would limit its application on next generation operating systems. A router may provide advanced security measures, but it cannot be fully customized for the systems that connect to it. Individual pieces of hardware are more customizable, and since traffic patterns vary greatly between users, this allows for independence from other user traffic patterns.

### **1.8 Implications**

Monitoring statistics of incoming and outgoing node traffic is much more beneficial than monitoring the network as a whole. Each user's traffic patterns can be monitored and provides a more detailed analysis of activity to and from a machine. This technique is not as beneficial for static worms and viruses, since detecting them is simply a matter of matching signatures.

Dynamic worms and viruses, however, find it more difficult to gain access to a system. A dynamic worm must enter the network and monitor the incoming traffic to a specific system. It would not be able to monitor the network traffic as a whole, since machines within a switched network do not see the same traffic. The worm must have a specific target and be more advanced to blend with the traffic to and from the node. In most router-based detection systems, only incoming traffic is monitored. With the proposed plan, outgoing traffic from a system is also monitored. Therefore, a dynamic worm would have to blend with the incoming and outgoing traffic to be undetected.

Since matching and monitoring would be performed in external reprogrammable hardware, the end system would not have to use any of its memory or processing power

exclusively for security. This allows more resources to be used for local computations rather than on network traffic analysis, as in a software based detection system.

## **1.9 Thesis Overview**

In Chapter 2, fundamental components of computer worms are introduced along with their attack patterns, evasion techniques, payload methods, and future. Also included is current research on Honeypots for intrusion detection on networks. Chapter 3 covers hardware implementations in current research and architectural aspects of attack resilient networks. Chapter 4 describes the details of line labeling and using relaxation to analyze an image. These techniques are then applied to a computer network example. Future research topics, for network labeling and hardware implementations, are covered in Chapter 5. The final chapter presents the research conclusions along with the significance of this research.

## **II. Literature Review**

### **2.1 Chapter Overview**

This chapter explains the fundamental components of computer worms, along with their attack patterns, evasion techniques, payload methods, and future. Several worm simulations are provided to demonstrate how quickly a network can be compromised, with varying levels of security. In addition to worm techniques, current research using Honeypots for intrusion detection on networks is also covered. The intent is to identify a set of suspicious activities that can and should be appropriately monitored.

Our simulations, using the publicly available Symantec Worm Simulator [30], reveal that the worms exploit vulnerabilities at all levels: Single nodes, group of trusted-friends, firewalls, routers, print-servers, etc. The symptoms are different at each of these personalities, but combined there is only a finite number. In addition, it can be envisioned that a hardware system that prevents abnormal activity and unusual high rates of traffic will prevent the spread of a worm and ease the administrator's burden of disinfecting a larger number of systems, allowing more time to apply patches to fix security holes in the operating systems.

### **2.2 Description**

To further enhance network security, techniques from Honeypots are used to monitor network traffic and activity, detect intrusions, and protect systems. Reprogrammable hardware can perform these processes in a stand-alone system. Others have done work in these areas, but none have combined all these techniques into a single research topic. In section five of this research, these ideas and techniques, will be used to

propose a platform for future network security devices and a more effective way to prevent malicious attacks on networks. To appreciate why these security techniques should be combined though, understanding how a worm attacks and gains control of a system will be covered first.

### **2.3 Computer Worms**

Understanding how quickly worms spread and gain access to systems is fundamental to network security and determining how to prevent spreading. The rate at which a worm spreads in a network is much faster than a human can gain control of the same number of computers. Individual nodes cannot provide enough information to determine the rate at which a worm may spread. When the information from individual nodes is combined with the actions and statistics from others, a useful measure of the network state can be gained and how fast a worm is spreading can be learned. There are a number of features, which will be covered later, that can be measured locally. Analysis of these can help determine if an attack has occurred. Although these measures cannot be used to trigger an alarm for an attack, they can be used to verify if anything has already occurred and provide forensic value.

Since a worm is automated, the worm code determines which ports open, how the worm's payload is delivered, and once a host is infected, how to find more hosts. All of these tasks are performed at an extremely rapid rate until the entire network is compromised. Furthermore, worms are persistent and therefore a network's defenses must be strong with all computers on that network having strong defenses as well. One weak link in a network is all a worm needs to compromise the entire network, since

within the network most computers “trust” each other. Finally, even though an “intelligent” worm has not been created yet, it is only a matter of time before one that can learn and adapt to its environment is created. A worm has been described as [32]:

*“a program that travels from one computer to another but does not attach itself to the operating system of the computer it “infects.” It differs from a “virus,” which is also a migrating program, but one that attaches itself to the operating system of any computer it enters and can infect any other computer that uses files from the infected computer.”*

A simpler and updated definition is [22]:

*“An independently replicating and autonomous infection agent, capable of seeking out new host systems and infecting them via the network.”*

### 2.3.1 Basic Components of a Worm

In a worm system, there are usually five components: Reconnaissance, Attack, Communication, Command, and Intelligence [23]. A worm must at least have an attack component.

- **Reconnaissance:** To expand infection and add hosts, a worm network must seek out other network nodes to infect. Reconnaissance is responsible for gaining access to other computers by exploiting security “holes” of other systems. The steps a worm may take are similar to a single attacker using port scans and service sweeps. These methods can determine what services are running on the machine and sometimes what type of system is being scanned. Reconnaissance can also be performed in a passive mode. By monitoring IP stacks or analyzing applications instead of performing scans, a worm remains unnoticed, while still discovering key systems. After determining the system type and a corresponding vulnerability, the system can be attacked.



- **Attack:** Attack components are the most dominant parts of worms and the most revealing. Worms gain entry to a system using standard remote exploits such as buffer overflows, cgi-bin errors, or Trojan horse techniques [23]. Most worms attack a specific platform and operating system and look for a limited number of known exploits to reduce the size of the worm so as not to overwhelm the network. With increasing network speeds, larger worms can be expected to have more exploits and be able to attack multiple platforms.
- **Communication:** For worms to spread, they need some way to communicate or transfer information between nodes. Information about vulnerabilities or mapping information must be sent by hosts or from a central location to nodes performing reconnaissance. These communication channels can be socket based (server and client) and use a variety of transport protocols or even email. Non-socket-based channels can be observed with a network traffic monitor. This way, signals blend into the normal network traffic and are more difficult to detect. Another way for worm nodes to communicate is through a central location such as a Web page. Anyone connecting to that page may also be infected, expanding the worm network at the same time.
- **Command:** A worm's usefulness can be increased after being interconnected by a command interface (either interactive or indirect) [22]. Along with the communications component, a hierarchy of nodes can circulate commands and expand the host's intentions. A "back door" is also commonly used to regain control of the system, without having to perform reconnaissance and the actual attack again.
- **Intelligence:** The true potential of the worm is reached when the nodes act together. The intelligence component exploits the knowledge contained in the nodes, which can be organized into a database of sorts. This database could

be a list of IP addresses or hostnames. Information for the database can be built in a number of ways, such as via email upon infection, special packets, or by some service for worm nodes [22]. This intelligence network is important to the worm, but it is also a vulnerability that can be compromised if the database is found and eliminated.

*Analyzing these five components shows that they can all be monitored in at least a passive way.* Something is always being transmitted or even just read, either with scanning or passive measures. Since files or IP stacks are being accessed, this is something that is countable. The number of times these are read can be used to determine if a large amount of scanning is occurring, or if the amount is normal. As will be explained later, these components lend themselves to finite measurable events and can give indication of an attack. These measurements should be used judiciously though to prevent false alarms and develop a reliable system for detecting attacks.

### **2.3.2 Scanning and Attack Patterns**

How a worm is spread is a function of how aggressively it attacks. If a worm randomly scours the Internet, it will not be very efficient. The worm will likely not cause much damage or survive for an extended period. Too narrow of an attack will cause more harm to that network, but may not spread very far beyond it to have a large impact on the rest of the Internet. Typical attack patterns include random scanning, random scans using lists, island hopping, directed attacks, and hit-list scanning [22].

A worm node that randomly chooses a network to scan, usually chooses a network with a block of 65,000 hosts (a /16 network) or 256 hosts (a /24 network).

However, pools of addresses on the Internet tend to cluster between 128/8 and 220/8. To be effective, the worm must focus on hosts that are accessible and potentially vulnerable to its exploits. Random generation is more likely to pick a network that is lightly populated. For example, several class-A networks below 127/8 are virtually unused except for researchers studying Internet security patterns and traffic [22]. The final element that is important to a random scan is the use of a good random number generator. If a poor random number generator is used, some networks may not be scanned at all and of the networks that are scanned, the scan will not be evenly distributed. Worms that use the random scan technique include Ramen and Code Red I worms. While weak random number generators held these worms back, a properly implemented random scan can achieve close to total coverage of the Internet in a small period of time [22]. A drawback to this scanning method is it tends to create highly visible and noisy traffic, which reveals its presence.

Random scanning using lists is similar to random scanning, but the worm carries a list of numbers to aid in the generation of networks. The list contains known assigned address spaces. This improves the worm's efficiency by focusing on locations where hosts are most likely to be located. Furthermore, since analysts often use unused address space to detect and track worms, a worm that avoids these spaces will be undetected longer. If the network is empty or only has a few hosts, the worm will still scan all the addresses but would be limited by timeout values based on failed connections. This can severely slow a worm's spreading progress. This preset list of numbers can also be used to predict the spread of a worm if the list is widely known.

Island hopping is another way a worm scans a network. Once all the resources are consumed in a network, the worm would move to the next “island” or random network block. This method has been very effective by implementing a weighted scan. About 50% of the time, the local network is scanned, 37% of the time the scan would move to a larger network that encompasses the current one, and 13% of the time a random network is chosen. Worms using this technique include Code Red II and Nimda, which lasted on the Internet for up to eight months [22]. These worms achieved a high degree of network penetration and exploited the trust policy in the networks. Since these worms usually exhaust a single network and then start to spread, when they are caught soon enough, they can be contained to that network and ultimately stopped. One possible way of determining if island hopping occurs, is by a node remembering the most recent 100 packets and the time between them. The last 10 subnets accessed would also provide valuable information. Then analyzing the modality of the list of most recent packets sent to nodes, would determine if a set number of connections are always occurring. If the modality of the list is close to one, it could be concluded that only a few connections are being made consistently. A modality of 100 would tell us that each connection has been to a different node each time. This could be an attack trying to spread itself to as many nodes as possible. Checking the modality of the subnet list would also help determine if island hopping is occurring and to which subnet it is happening the most.

Another type of scanning and attack is directed at a particular network. A worm of this type could be part of a focused information warfare effort to penetrate a certain network for information or to simply compromise the network’s security, causing

services to be disabled as preventative measures. Worms of this type can attack as soon as it is introduced, which would catch the target network off guard. However, the strength of the worm has not been built up and therefore can be easily filtered by IP address. The other type of worm has a waiting period which allows time for other more vulnerable networks to be compromised. Then, those networks can work together to attack the particular network of interest. This type of worm may have the size and strength it needs, but while it was building resources it may have been identified and defenses developed.

The final method is hit-list scanning, where an initial scan of nodes determines which ones are most vulnerable. A list is generated which contains the addresses and information of the nodes vulnerable to the worm, which focuses attack. Once compromised, a child worm gets half of the list and the other half stays with the parent [22]. With each iteration, the worm's efficiency improves. One of the problems with this type of attack is with the initial scan. Since the worms are looking for a particular weakness, this can be detected on network monitors and alert the community. Further, as the worm moves through the network it may eventually use up all network bandwidth. However, if the hit list were sorted so that larger bandwidth networks were hit first, network congestion could be reduced.

Once again, scanning and attack patterns perform specific types of activities that can be observed and quantified. For example, with island hopping, the destination of most recent packets and the time between them can determine if an attack is occurring. This technique would also work for a directed attack, but the measurements would be

different. The most recent subnet list would not change, but the rate at which connections are made would most likely be larger than usual to a particular network. Detecting a random scan would have characteristics of a quick connection rate, but the destinations of packets would be larger than usual, but still have a finite set. Specific subnets would not be destinations either, since unused address spaces would purposely be avoided.

### **2.3.3 Payload Methods**

Once a worm has gained access to a target system, there are several ways to deliver the malicious code to the newly infected node. The first is direct injection, but this method is easily blocked using a firewall on unused ports for most connections. With a firewall, the initial connection would likely have been blocked as well. If the connection is not blocked, direct injection can reuse the connection to deliver the payload and no other services need to be setup on the system. This reduces the complexity of the worm and masks its presence by not invoking additional services.

The second method is a child to parent request. Suppose a child worm has access to a node; the child can request that the parent send the worm payload. Code in the parent node then acts as a small server, which fulfills the requests of the child worms. Similar to the direct injection method, these so-called back propagation [22] requests can be blocked using a firewall or NAT (network address translation) device that does not allow connections to the parent network.

Finally, a central source delivery method updates and adds new capabilities to worms easily. A single location, such as a web site, can host all the files of the worm. Changing any of those files affect future instances of the worm. If the files are compromised, the life of the worm can be significantly reduced. Compromise may occur if the site is be shut down or corrupted files replaces the current files, causing the worm to destroy itself.

#### **2.3.4 Future of Worms**

As embedded devices depend more on network connections, worms will not be limited to traditional desktop computers and their operating systems. Embedded devices and appliances will also become targets. Many of these devices have services to configure them and there may be programming errors in these services or fundamental security flaws in the configurations. These devices include cable/DSL modems, routers, switches, or even appliances such as network printers and cell phones. Almost any device that can connect to the Internet will be a target. The attack may not be directed at these devices necessarily, but they could be used as storage devices for the worm's files or to divert attention away from the primary attack on a network.

Worms usually only attack a single type of system and use the same method to spread between machines, or in other words are mono-morphic. Static worms, although simple to implement and effective, are becoming less effective due to antivirus programs and other network devices which can detect static worms. "Intelligent" worms, however, can learn how to enter a system and hide. The Samhain projects purpose is to design this

more effective Internet worm. Samhain lists seven requirements and guidelines for a more powerful worm [38]:

1. Portability across operating systems and hardware architectures. If a worm can exploit vulnerabilities in each type of system, the number of targets can be maximized.
2. Invisibility from detection. A worm than can hide has a much better chance of being able to spread and not endanger the worm network.
3. Independence from manual intervention. While the worm already spreads on its own, it should also be able to learn on its own and adapt to its surroundings in the network.
4. Automatic updates. Besides learning new techniques, worms should also be able to update their own database of exploits.
5. Integrity of the worm host must be preserved. If a worm is detected, it should not allow its executables to be examined by others.
6. Avoid static signatures. Polymorphism allows a worm to avoid signature-based detection methods.
7. Overall worm net usability. The worm's network should be able to focus and achieve a specific task.

Even if such a “super” worm, were created, size is still an issue. While network speeds are increasing, a worm with large files and overhead will prevent itself from spreading throughout a network. No worm yet has incorporated all these methods, but some can be found in more recent worms such as variants of the Slapper worm. These worms use common web server process names such as httpd to hide. An updatable worm called the Windows Leaves worm retrieves updates distributed on the Internet. The damage and chaos such a worm could cause is easily imagined. The next section discusses several



current worms and provides examples of how quickly they spread throughout a simulated network.

## 2.4 Worm Simulation

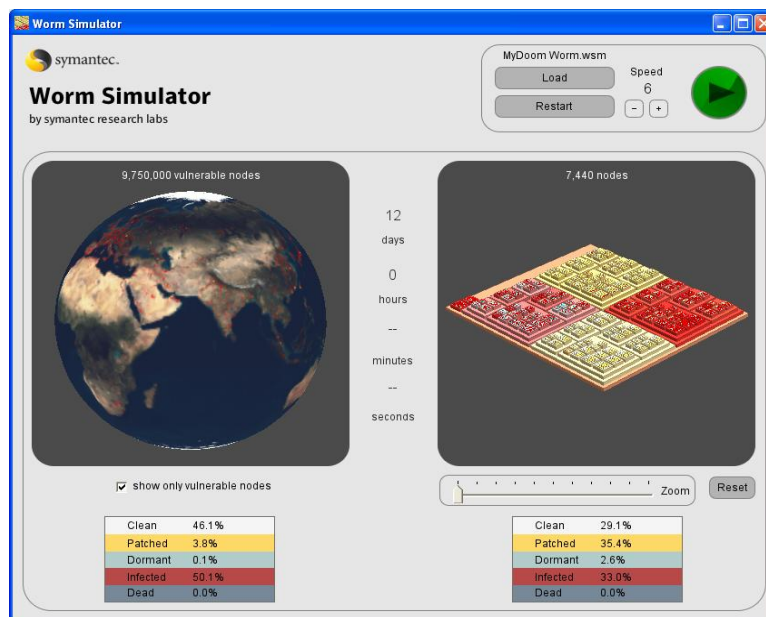
Several worm simulation studies were performed using a publicly available tool called the Worm Simulator, from Symantec Research Labs [30]. Worm Simulator allows a user to graphically view the spread of six different worms, the rate of infection, and the time for the worm to reach its maximum potential before systems are either cleaned or patched by network administrators. This tool shows how quickly a worm can spread and on average how long it takes for a network to be patched against a worm's exploit. Of the six worms available in the simulator, the Blaster, MyDoom, Sasser, and Slammer worms were selected. In the simulations of each worm, the networks are divided into four equally sized sections. For reference purposes, I have defined each square as a quadrant and the relative position in the network using the north, south, east, west directions. Therefore, the north quadrant is the top network in the figures. The simulator demonstrates how quickly a worm will spread in four similar networks with differing security policies. These policies include:

1. **No security** (East quadrant) - Almost all of the nodes are vulnerable to the worm, and machines are patched very slowly.
2. **Only firewall security** (West quadrant) - The network is protected by a perimeter firewall, but a small number of users are allowed to connect by VPN, which is not protected by the firewall. In addition, a high number of systems in this network are vulnerable and machines are patched slowly.
3. **Strong host security and network security** (South quadrant) - Similar to the previous network except it has a better internal security policy and fewer of the

nodes are vulnerable, many machines are patched against the worm before they are infected, and most of the machines are patched quickly after they are infected.

4. **Only host security** (North quadrant) - This network has the same solid internal security policy as the third network, but does not have a perimeter firewall. Only a small percentage of the nodes in this company are vulnerable. Although, since the network has no firewall, nodes will initially become infected, but patching of uninfected nodes is fast and patching of infected nodes is even faster [30].

In Figure 2, a screenshot of the Worm Simulator portrays these four subnets and the overall percentage of infection for the network as a whole. The amount of time that passes is also displayed, so that an idea of how rapidly a selected worm will spread can be seen and when system patching has been completed.



**Figure 2: Screenshot of Symantec Worm Simulator**

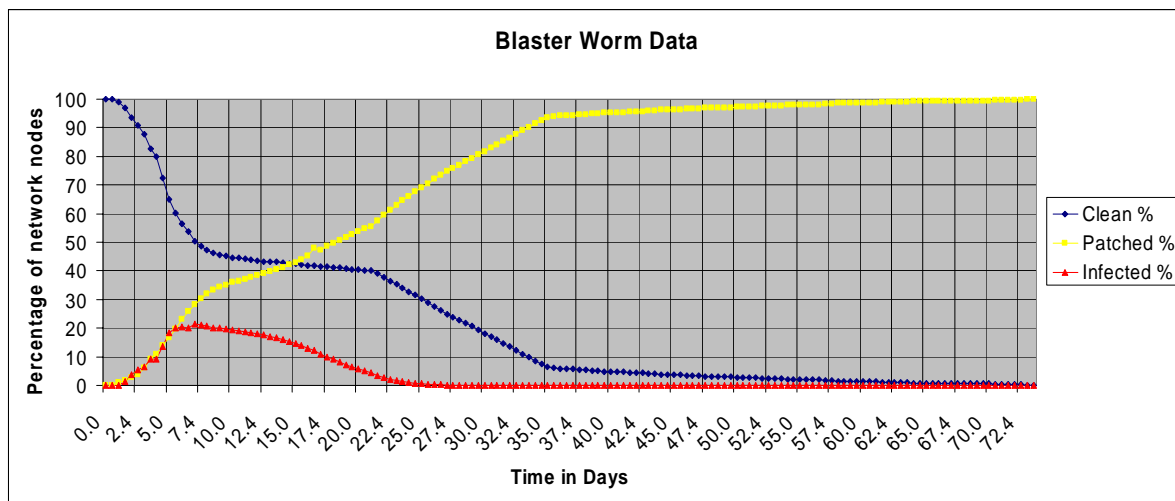
*In screenshot is to demonstrate how information about the worm is presented to the user of the publicly available program. For this particular example, the MyDoom Worm was selected from the six preset datasets and allowed to run. The simulator illustrates that after twelve days, 33% of the 7,440 nodes are infected among the four networks. The two, less secure, networks have mostly been compromised, while the other networks have continuously applied patches to vulnerable systems.*

### **2.4.1 Blaster Worm Simulation**

The Blaster worm exploits the DCOM RPC (Distributed Component Object Model of the Remote Procedure Protocol) vulnerability using TCP port 135 on Windows 2000 and XP machines. Even though Windows NT and Server 2003 machines are also vulnerable to this exploit, the worm is not designed to replicate in those operating system environments. When a machine is compromised, the worm attempts to download the msblast executable file into the system32 directory and execute it. It then attempts to perform a DoS attack on the Microsoft Windows Update Web server, to prevent users from applying patches to the computer. Additional features of this worm include checking to see if the system is already infected, and can generate additional IP addresses. The creation of the IP addresses are determined by an algorithm where 60% are randomly generated and the other 40% are made to infect other systems in the current subnet. This worm also has a payload trigger so the executable is run if the date is between the 16<sup>th</sup> and the end of the month or if the date is between August 16<sup>th</sup> and December 31<sup>st</sup>.

Several key events occurred during the worm's lifespan and the patching process. Initial infection from a home VPN connection occurred at hour 22, where only 0.9% of machines had been patched. After one day, almost the entire home network was compromised and a subnet within a corporate network has been engulfed. The worm has also spread to several nodes of another network as well. By day six, peak infection was reached in the firewalled only network, while other networks have the attack under control. After one month, all infections were removed. According to the worm

simulator, over three months later, all four network's systems have finally been patched. In Figure 3, the percentages of network infection and patching rates have been acquired from the simulator and can be seen for the 1,849 nodes simulated. Under a week's time, the worm reached its full potential. Only networks that applied patches regularly and maintained the best security policies remained reasonably secure. Even some of the systems with the "best security policy" networks still managed to be infected and were allowed to spread the worm.



**Figure 3: Blaster Worm Cleaned, Patched, and Infected Rates**

*From this data, it can be seen that the initial infection rate of the Blaster Worm was similar to the patch rate. Therefore, if the worm had not leveled off in spreading the infection, half of the systems may have been infected, while the other half was patched. The important part of this graph is how long it took to fully patch all the systems. It took around 80 days, to apply the patch to all systems. In this time, if the worm could have entered into another network, there may have been a second wave of infections.*

## 2.4.2 MyDoom Worm Simulation

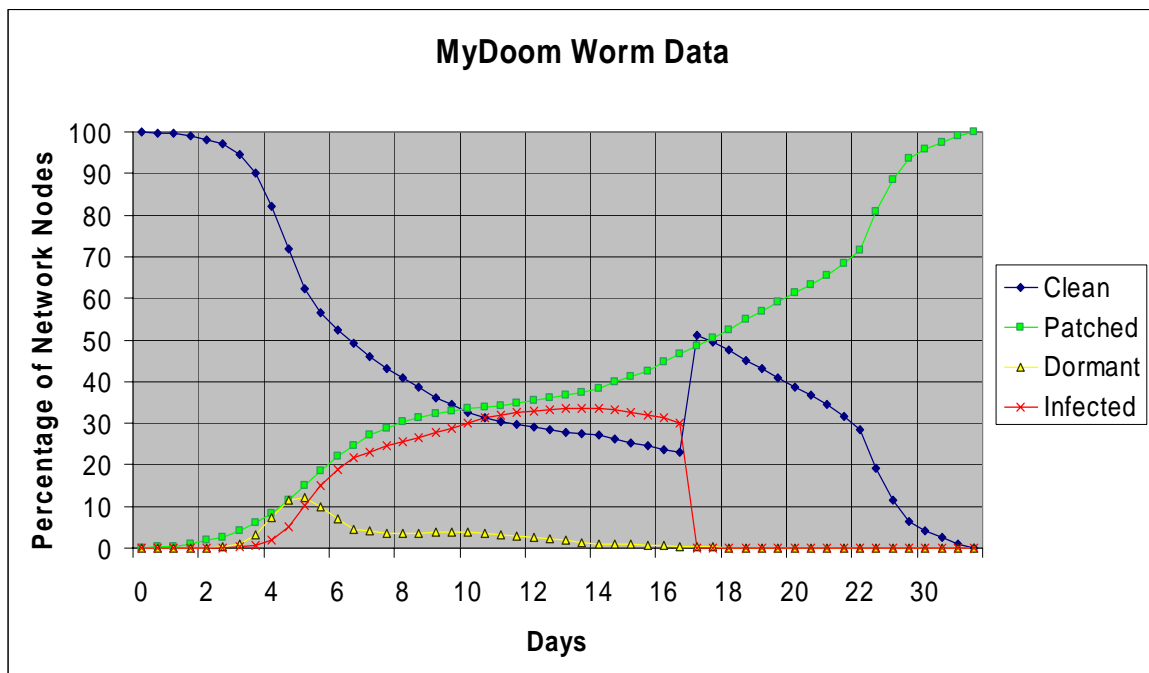
The MyDoom worm is not so much an operating exploit, as it is a social engineering exploit. Users receive an email with an attachment containing the worm, and

when opened the worm reads the email addresses of a file and mass-mails itself to the recipients. File attachment extensions include: .bat, .cmd, .exe, .pif, .scr, or .zip. After the computer is infected, the worm sets up a backdoor using TCP ports 3127 through 3198. These open ports allow an attacker to connect to the computer and use it as a proxy to gain access to its network resources or download and execute arbitrary files. In addition, the infected system has a 25% chance that the worm will perform a Denial of Service (DoS) on February 1, 2004 starting at 16:09:18 UTC. If the worm does not start a DoS attack, it will not mass mail itself. It has a trigger date to stop spreading and performing DoS-attacks on February 12, 2004. After this date, the backdoor component still continues to function and allows access to attackers.

In the simulation, both networks with lower security policies became almost fully infected with the worm. Networks that had constant attention from administrators and updates did not have many infections of the worm. This shows that if enough attention is given to a network, even if systems become compromised from worm activity, there is a chance to prevent deep infection. A goal of this research is to devise a network scheme that becomes self aware of infections or malicious activity so that administrators do not have to constantly watch network traffic and disinfect systems when an outbreak occurs. Even though the two more secure networks did not become overrun with the worm, network administrators played an active part in patching and removing infections from the systems in the non-overrun networks and did not use automated tools.

Figure 4 shows that initial infection occurred on day two, and on day five the percentage of dormant nodes reached a maximum. The worm was first spread to a

number of computers, while other systems contained the worm in a dormant state. These systems most likely contained the worm in an email they received, but the attachment had not been open and executed yet. The maximum percentage of infection was reached on day fourteen, when the two less secured networks had a majority of their nodes compromised and dormant nodes had mostly been activated or patched. On day seventeen, which represents January 26<sup>th</sup> 2004, the trigger date was reached and all infected nodes stop spreading and returned to a “clean” state, waiting to be patched. Even though those systems still had a backdoor open, the worm did not spread itself, nor continued to infect other machines, so the worm and infection was considered inactive.



**Figure 4: MyDoom Cleaned, Patched, Dormant, and Infected Rates**

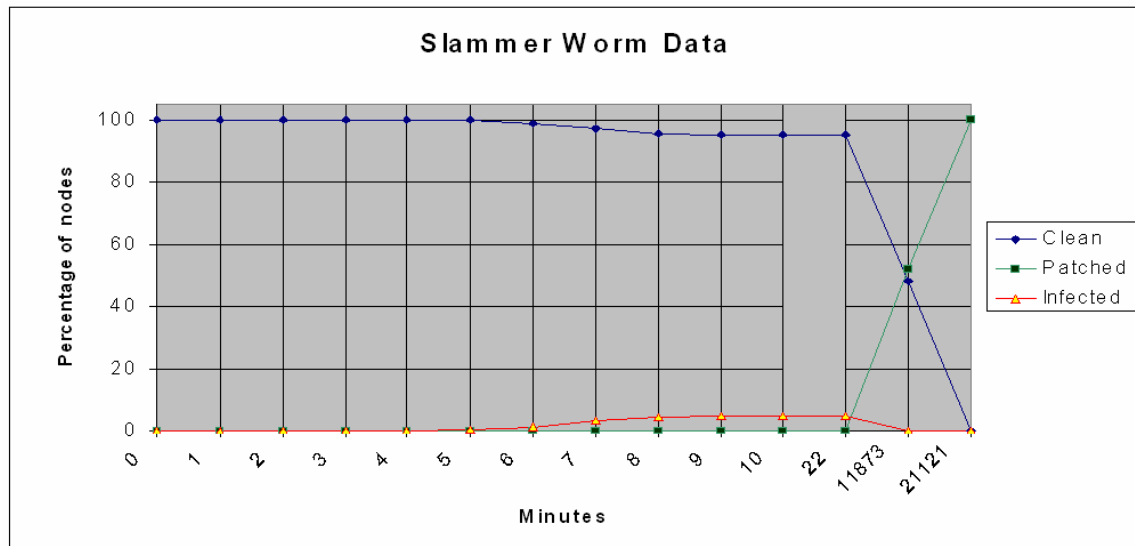
*The information gathered from the worm simulator shows that initially about one-tenth of nodes are: infected but in a dormant state, infected and spreading the virus, and one-tenth have also been patched. This leaves 70% of nodes untouched so far. After day five, which represents February 1, 2004, these dormant nodes start the DOS attack and other infected nodes continue spreading until February 12<sup>th</sup>, represented by day seventeen. On this day, all attacks stop and infections are considered “clean” until patched.*

### **2.4.3 Slammer Worm Simulation**

The Slammer worm demonstrates how quickly a worm can spread before any detection or patching can occur. Even though this worm only targets systems running Microsoft SQL Server 2000 and Microsoft Desktop Engine (MSDE) 2000, it searches for these services by continuously sending out traffic to randomly generated IP addresses in an attempt to make a connection. By doing so, it performs a DoS attack on a network due to the large number of packets it sends out. In the simulation of 500,000 nodes, only 75,000 are vulnerable to this worm throughout the world. In the more focused example of a company network of 1,850 nodes, many of these vulnerable nodes are infected within a matter of seconds. The rest of the vulnerable nodes are not infected because the worm itself causes so much network traffic, preventing itself from spreading. Furthermore, the random generations of IP addresses do not cover the entire network in that time.

Two minutes after initial infection, the simulator showed that 25 of the 1,849 nodes had already been infected and one minute later 25 more are infected. Nine minutes into the simulation, the worm reaches its maximum potential and an entire small network is infected. The problem with this rapid spread is that it consumed all network resources and choked itself, limiting the spread of infection to roughly 100 nodes. Patching did not even start until eighteen minutes after the initial infection and the network took several days to become fully patched. The data in Figure 5 shows how the infection spread in a matter of minutes before any patches were applied. If the worm had not sent out so much

data, which caused enough network congestion to prevent its own spread, other networks may have been infected before patching was initiated.



**Figure 5: Slammer Worm Cleaned, Patched, and Infection rates**

*This worm, unlike the previous ones shown, reaches peak infection in minutes rather than hours or days. The drawback to this technique is it consumes its own network bandwidth too quickly and prevents the spread to other networks outside of its own domain. Patching is not even half way complete until day eight. The overall network is fully patched around day fourteen.*

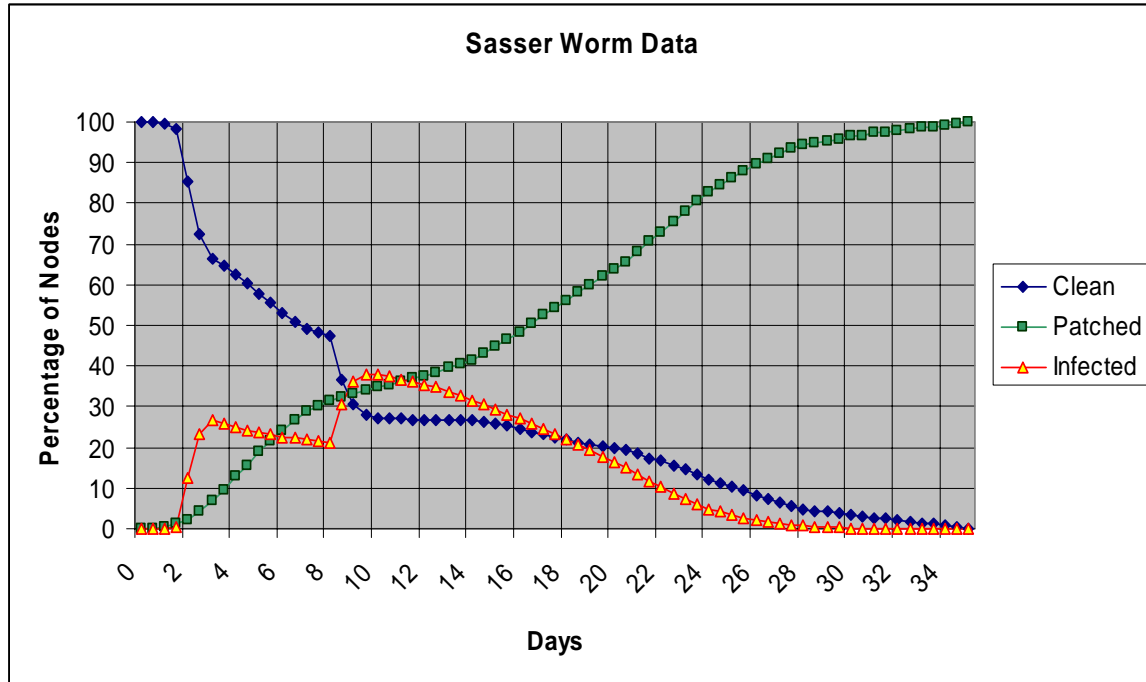
#### 2.4.4 Sasser Worm Simulation

The Sasser Worm is investigated because it scans random IP addresses for systems with the LSASS exploit, found in unpatched Windows XP and 2000 machines. After the worm has found a system it can connect to and exploit, the worm attempts to create a mutex to see if the system is already infected. If the system is not infected, it adds itself to the RUN key of the registry so it will run on each start-up of the computer. The worm uses the AbortSystemShutdown API to prevent the computer from being shutdown or restarted. To spread the worm to other hosts, it creates an FTP server on



TCP port 5554 and uses an algorithm to select the IP addresses it will infect next. The IP address is completely random 52% of the time, 23% of the last three octets of the IP address are changed, and 25% of the time only the last two octets are changed to random numbers. Since this worm uses 128 threads, the system it infects will dedicate most of its CPU time to the worm and become unusable by the user. Although older Windows operating systems (95/98/ME) cannot be infected with the worm, the worm can run on these machines and infect other systems.

Simulation of this worm showed it infected one network completely, and then hours later, infected another because of its random IP address generation. The Sasser Worm spread very rapidly once it had access to a network. Within two days, the least secure network is fully compromised, while other networks began patching procedures. Seven days later, while some systems had been patched, the initially compromised network infected another system on a network, which had been overwhelmed with infection as well. The initial compromised network did not receive a significant patching procedure until day twenty, where the second network was still moderately infected. This can be seen in Figure 6, the percentage of nodes infected exceeds the patched nodes twice. The initial infection was slowed by an increase in patched nodes, but the second network infection happens rapidly and exceeds the number of patched systems again, until administrators catch up and apply patches to more systems. If the other two networks did not have higher quality security policies and apply updates on a regular basis, these networks would have also been compromised deeper and much quicker by the two other infected network generating random IP addresses to spread the worm.



**Figure 6: Sasser Worm Clean, Patched, and Infection rates**

*Two waves of infections occur with the Sasser Worm in this simulation. The first wave is when one of the less secure networks is fully compromised. Some patching of this network occurs before the worm is spread to the next network, but not enough to prevent another outbreak. The graph shows that the infection rate greatly outweighs the rate at which patching occurs. If all the networks had a slow patching method, one would expect to see four separate peaks in the graph, showing that the percentage of nodes increased drastically each time the worm spread to a new network. Eventually though, the more secure networks were immune from the worm, and the other networks began patching procedures to bring the outbreak under control.*

With a network anomaly detection system, the rate the worm spread to the second network would have been decreased drastically, since the first initial connection generated a rapid spread in the network once a machine was compromised. In a network such as these, machines do not connect to other LAN machines on a regular basis. When this irregular action occurs, the anomaly detection and prevention unit would prevent this

attack from spreading by recognizing the irregular connections and the rate the systems try to connect with others.

## **2.5 Honeypots**

Honeypots are distinctly different from firewalls and intrusion detection systems [18]. They do not solve a specific problem; “they are a highly flexible tool whose value lies in unauthorized or illicit use of that resource [29].” This is a general definition, since almost all honeypots work the same. In an ideal network, a honeypot will not see any traffic and therefore, is not productive to the network. In a real world network though, any interaction with a honeypot is almost certainly unauthorized or malicious activity. Applying this to a self-aware network would allow the honeypot to alert the other neighboring systems of the vulnerability, since the compromised system would fail to update its own state. To further increase the value of honeypots, several honeypots on a network, which are aware of each other, could communicate periodically to attract attention to themselves. This would increase the chances of malicious content noticing these “less secure” systems and attempt to compromise them.

### **2.5.1 Advantages and Disadvantages**

Even though honeypots are simple in concept, they are a valuable asset and have many advantages, such as collecting small data sets of high value, capturing new tools and tactics, and using minimal resources. Instead of collecting several thousands of alerts a day, a honeypot only collects small amounts of critical information. This information

can then be analyzed further for security detailed security threats. Analyzing this data is much easier and cheaper, since common traffic information has not been collected.

The design of a honeypot allows it to capture all data sent to it. Since there are no complex algorithms to develop or signatures to update, older hardware with minimum specifications for the operating system can handle an entire class B network. Because the concept and technology is simple, there will be less of a chance for configuration errors. That is also why this technology works well in encrypted or IPv6 environments, unlike other security technologies such as intrusion detection systems.

Like all technologies, there are disadvantages, and that is why honeypots need to work with existing security measures. Honeypots capture all activity that it encounters, so unless the attack interacts directly with the honeypot, no information captured and analyzed. Another risk of deploying a honeypot is the possibility of it becoming compromised and used against its own network. “Depending on the type of honeypot deployed, the risk could be no more than the risk of an IDS sensor failing [29].” How large the risk of being compromised, comes down to what category of honeypots will be used.

### **2.5.2 Types of Honeypots**

There are two general categories of honeypots [29], low-interaction and high-interaction honeypots. Low-interaction honeypots emulate services and operating systems, which prevents full interaction. This emulation can range from just a physical

appearance, such as an FTP login listening on port 21, or a more interactive emulation with a variety of additional FTP commands. Low-interaction honeypots have minimal risk and are simple and easy to maintain. Specter, Honeyd, and KFSensor [29] are some examples of low-interaction honeypots that allow easy selection of the operating systems and services you want to emulate and monitor. These programs never allow access to the operating system, so the attacker cannot use the system against the network. “The main disadvantages with low interaction honeypots is that they log only limited information and are designed to capture known activity. [29]” Furthermore, it is easier for an attacker to detect a low-interaction honeypot. A skilled attacker will eventually detect their presence.

High-interaction honeypots, such as Symantec Decoy Server and Honeynets, are systems that are more complex. Instead of emulating services, they use real operating systems and applications. This allows the attacker to use all their skills and tools as if they were attacking an actual productive system. The knowledge and information gained from monitoring the attack, gives one a better understanding of the attackers intents. Since a high-interaction honeypot is an open environment, it captures all activity, even unexpected behavior. For example, “honeynets have captured encoded back door commands using a non-standard IP protocol (specifically IP protocol 11, Network Voice Protocol). [29]” However, high-interaction honeypots also increase the risk of attackers using the real operating system for attacks. To prevent this, fail-safe measures must be in place to stop an attack or disconnect the machine from the network, inhibiting other productive machines from compromise. In most cases, high-interaction honeypots can do

anything low-interaction honeypots can. The benefits of using a high-interaction honeypot to collect more information must be weighed against the complexity of setup, and the risk to the entire network if compromised.

### **2.5.3 Value of Honeypots**

In general, there are three ways a honeypot can prevent network attacks. “Sticky” honeypots, such as the LaBrea Tarpit [29], are low-interaction and can help defend against automated attacks by slowing scan rates, potentially even stopping them. These automated attacks usually randomly scan entire networks searching for vulnerable systems to take over and use to spread themselves. By interacting with the attack, sticky honeypots slow the attack down. One TCP trick that is used to slow an attack, is setting the windows size to zero [29]. Honeypots can also be used confuse an attacker and waste their time and resources on a non-productive system. In the meantime, the organization can detect this activity and respond to the attack or try to trace them. On the other hand, if the use of honeypots is made available to the community, then this may deter attackers since they would be unsure of if a system is worth attacking.

Detection is the next critical purpose of a honeypot. In any establishment, there is bound to be some type of loophole or failure in the system. By quickly detecting and reacting to an attack, the damage caused can be sustained. With other technologies, such as logs or IDS, they generate too much data for one to sort through and determine if an attack is currently happening. For efficient and quick detection, low-interaction honeypots make the best solutions.

Response is the final important value of a honeypot. “Production systems, such as an organization's mail server, are so critical that even though it has been hacked, security professionals may not be able to take the system down and do a proper forensic analysis. [29]” With this in mind, leaving a system up and running continuously pollutes any data on that server used to analyze the attack. A honeypot though, can be taken offline anytime a forensic analysis is needed, since there it has no productive value. High-interaction honeypots are most useful for response since they collect the most detailed information of an attack.

Honeypots, although simple in nature, are an important part of gathering information on attacks and should be integrated into today's networks. “For centuries, military organizations have depended on information to better understand whom their enemy is and how to defend against them. Information security is no different [29]”. Honeypots address this need. Either by monitoring a company's network space, or large unused address spaces on the internet, the information collected can help us gain knowledge and understanding for future attacks.

## **2.6 Summary**

Fundamental components of computer worms, along with their attack patterns, evasion techniques, payload methods, and future have been discussed in this chapter. The worm simulations performed reveal that worms exploit vulnerabilities at all levels of security. Secured networks are vulnerable until fully patched, since some worms use social networking to propagate themselves. Research using Honeypots is also introduced to identify a set of suspicious activities that can and should be appropriately monitored.

These activities may seem like an infinite set, but through simulation is determined that there are only a finite, albeit large, number of specific types.

In the following chapter, research work has shown that attacks can be delayed, if not prevented all together in certain situations, using a combination of hardware based intrusion detection systems and network monitoring. Devising a hardware based network monitoring device allows all processing of packets and analysis of those packets to be performed allow an end-user's system to be easily and reliably quarantined from the rest of the network. When an operating system controls the monitoring of its own security services, this leads to a single point of failure. Once the system is compromised, any feedback from that system cannot be trusted. Without external monitoring and detection, a system's self-analysis of security using software proves nothing. Only a hardware solution either built into the system or external to the system that can monitor its activity, can determine if the system is behaving abnormal and has been attacked.



### **III. Architectural Elements of Attack Resilient Networks**

#### **3.1 Chapter Overview**

Hardware solutions for network security are starting to appear more frequently in today's modern networks. Some of these solutions are similar to host based virus scanners, searching for a fixed signature in traffic patterns. When viruses become dynamic in nature though, these methods will not be reliable. Existing methods involve some kind of periodic updates. Usually the local agent is responsible for keeping systems current. Even these databases have a latency; for a good reason. Substantial improvement can be made if the update can be augmented by a decentralized means. Our relaxation based framework makes it possible.

Attack resilient systems allow users of a network to continue normal everyday tasks, other than the functions that have become disabled, while only disrupting the nodes that are violating normal protocol. To a limited extent, nodes are being equipped with attack resilient compatibilities using software agents. For example, anti-virus with updates at every reboot and in conjunction with forced reboot. There is a drawback in such an approach, that the software agent has to run on a potentially infected node. One way to strengthen the guarding process is to use hardware. Contemporary research on hardware based solutions is covered in this chapter along with components that could be used towards an attack resilient network. These hardware systems have two underlying advantages over software measures: isolation and partitioning of the sentry while protecting the node and resource allocation. More on attack resilient systems will be

described in this section, but with the bulk of security removed from the node, the node's resources can be used for other tasks or simply accelerating common tasks.

### **3.2 Hardware-based Prevention and Detection**

Hardware-based threat detection and analysis has been the subject of modern research. Hardware, such as a Network Intrusion Detection System (NIDS) has been used to string match against incoming packets. These devices compare the incoming packet payloads to signatures of hostile data. Research has shown that hardware or FPGA (Field Programmable Gate Array)-based string matchers out-perform a software-based system by up to 600 times for large patterns [13].

Common NID and prevention systems use predefined signatures to search network traffic. Since string matching is the most computationally expensive step of the detection process, NIDS' only apply this technique to the most suspicious packets. A popular NIDS is Snort ([www.snort.org](http://www.snort.org)), which checks port numbers, packet headers, and flags to ensure a given packet has a high likelihood of containing malicious data before performing string matching. Software scanners are not fast enough to monitor traffic with the resources provided by the system. Hardware can make use of parallelism to perform deep packet inspection with high throughput [20].

Implementing a system that scans the full payload of packets presents several challenges. First, the location of a targeted signature in a packet payload could appear at any position in traffic, so if a string matcher is not intelligent or only starts matching at the beginning of a packet, malicious code can go undetected. Furthermore, signatures could span multiple packets or be interleaved among multiple traffic flows. String

matching would have to keep track of the packet flow along with the sequence of the packets to reconstruct the flow and match the fragmented code to a signature [9]. Contemporary research has provided a means of meeting some of these challenges, by keeping packets in a queue and re-ordering them when the last packet has been received. Research work which has started to solve some of these problems is described in the next section.

### **3.3 Relevant Research**

Relevant research, in the area of virus detection with hardware, includes work with FPGA-based string matching, an open source NIDS called Snort, that provides test data for performance comparison, and a Java based hardware design tool kit (JHDL) used to implement a module generator. Current research work in string matching has been performed by Sidhu and Prasanna [13] to accelerate grep (a command used to find lines in file(s) with particular text or regular expressions) searches with FPGAs. Their compilation strategy quickly converts a regular expression into an FPGA circuit. The regular expression is then compiled into a Nondeterministic Finite Automaton (NFA) and then directly implemented with FPGA hardware. In software methods, a Deterministic Finite Automaton (DFA) must be derived from the NFA. This hardware method decreases compilation time and simplifies the process, especially as the regular expression increases in size. Table 1 and Table 2 show the comparisons of performance for these two methods. As shown, when the regular expression size and data transfer size increases, the FPGA has the clear advantage over the software implementation. For the hardware method, the largest sized regular expression throughput improves between 300

and 500 times over the software method, depending on data transfer size. Both methods were executed on a Pentium III (750Mhz) computer running Redhat Linux 2.7.3-10. The hardware circuit was implemented on a PCI-based board with a Virtex XCV1000 FPGA, housed in the computer.

**Table 1: FPGA vs. software regular expression performance on 1MB data set sent in 1kB chunks [13]**

Size of Regular Expression (# of non-Meta characters)	Hardware Latency (ms)	Software Latency (ms)	Hardware Throughput kB/s	Software Throughput kB/s	Hardware CPU Utilization	Software CPU Utilization
47	< 1	< 1	390	432	33.9%	11.2%
435	< 1	3.2	340	197	33.6%	67.6%
844	< 1	37.6	381	23.5	34.3%	91.9%
1420	< 1	104	284	8.9	28.4%	96.3%
2689	< 1	240	291	4.63	24.7%	98.3%
4971	1.2	970	331	0.99	43.7%	99.4%

**Table 2: FPGA vs. software regular expression performance on 16MB data set sent in 16kB chunks [13]**

Size of Regular Expression (# of non-Meta characters)	Hardware Latency (ms)	Software Latency (ms)	Hardware Throughput kB/s	Software Throughput kB/s	Hardware CPU Utilization	Software CPU Utilization
47	< 1	< 1	862	884	19.3%	11.8%
435	< 1	50.9	870	278	18.1%	97.3%
844	< 1	602	824	24	16.3%	98.3%
1420	< 1	978	826	14.9	19.3%	99.6%
2689	< 1	1930	838	7.58	20.1%	99.6%
4971	7.38	8400	784	1.72	38.5%	99.8%

Snort examines network traffic, and logs intrusion events. Since it is open source, rules can be added or deleted to improve detection or performance relatively. Snort provides the rule-set, which a module generator extracts and creates a regular expression to match extracted strings. Since Snort is updated with rules, the circuits must be

optimized to make use of the resources available on the FPGA while still matching as many patterns as possible.

Another tool used in their research was JHDL (Java Hardware Descriptive Language). JHDL is a Java-based design tool consisting of a set of Java libraries to perform programmatic structural design. Circuits are created by calling constructors for the corresponding JHDL object and passing wire objects as constructor arguments that are connected to the ports of the circuit. After connection, these circuits are verified with the JHDL simulator and an EDIF (Electronic Design Interchange Format) net-list created for Xilinx place and route software.

Other related research includes using programmable logic devices (PLDs) to perform the regular expression matching. A system using three components has been implemented at the Applied Research Laboratory [20], in Missouri, to protect networks from Internet worms and virus attacks. The system is comprised of a Data Enabling Device (DED), a Content Matching Server (CMS), and a Regional Transaction Processor (RTP). These components scan every packet that enters and exits the network. Signatures are kept updated by an administrator via a table on the CMS, this in turn programs the DEDs to scan for the new signatures. In the DED, packets are scanned with the FPX (field programmable port extender) and processing is done on a Xilinx Virtex XCV2000E FPGA. “Layered protocol wrappers parse the headers and payloads of packets using high-speed circuits implemented as combinational logic and state machines in the FPGA.[20]” To take action on malicious activity the DED contacts the RTP, which after consulting a database, determines the appropriate action the DED should

take. DEDs are installed at aggregation points so that traffic flows through at least one of these devices before being routed to other networks or the Internet. A single RTP can administer up to 100 DEDs and long, distinct strings are used so false positives are minimized.

This system, using finite automata to scan for regular expressions and Bloom filters for fixed length strings, can identify up to 10,000 unique fixed-length strings with a bloom filter on a single FPX card. The number of expressions searched grows approximately linearly with the amount of the FPGA logic on the device. Both methods used by the system can process data up to 600Mbps. With four modules in parallel, scanning speed increases to 2.4Gbps. The number of terms to be searched does not affect throughput as long as the set of matching signatures fit in the FPGA after being synthesized. While this setup only scanned traffic in one direction, the system could be modified to scan the content of traffic in both directions, which would allow the system to detect the presence of confidential or classified materials and block its release.

### **3.4 Potential Indicators of Malicious Activities**

Instead of scanning for signatures, another solution is discussed in “Throttling Viruses: Restricting propagation to defeat malicious mobile code,” [34] which slows the spread of viruses by placing network connections to new computers in a delay queue. This method is more tolerant to non-malicious non-normal traffic. False positives are delayed, but still tolerated, whereas malicious code or worms that connect at higher rates are penalized more. When a new connection is initiated, it is added to a delay queue, where it waits for the second system to process it. A new connection is determined by

comparing the connection to a short list of recent connections. The sensitivity of this system can be set by varying the length of the list to compare against. By implementing this method in an external network monitoring device between the user and the network, processes can be analyzed quicker and not use system resources. If users on both ends have a similar device, malicious code is slowed down even further as outgoing and incoming connections are both delayed. For false positives, this is a minor inconvenience compared to a fast moving worm that causes much more damage. This method best prevents fast spreading viruses or worms, such as Warhol worms, which try to make multiple connections in a very short time [22]. With slow moving viruses, this method will not be as beneficial, but it will still slow the spread. To be most effective, both the sending and receiving machines should have a device as described and the initial connection would be delayed at both ends. Continued connections would eventually be considered normal and permit a user to operate without delay.

### **3.5 System Behavioral Description**

An ideal attack resilient network would not need to interact with a user to prevent the spread of an infection. In the standalone hardware, anomaly detection/prevention, network monitoring, and status updating should be performed. Incoming and outgoing traffic should be monitored from a statistical standpoint, any traffic that does not fit into a users “normal” traffic pattern is flagged. The information to create these “patterns” could be retrieved from printer logs, network logs, remote application logs, network traffic based on port usage (e.g. e-mail, ftp connections, web browser, secure shell connections.), and file server logs. Additional information can be gained from other log

sources not related to the user's computer. Combining building access logs, user's roles in a company, telephone calls (VOIP), and other physical access methods, more detailed "typical" traffic patterns can be created for an individual [21]. The benefit to this approach, from the viewpoint of the user's system, is that monitoring traffic is operating system independent, and should not be easily detected by an outside source. If an anomaly is detected, the hardware can intervene and deny incoming or outgoing traffic. The device should also alert other nodes in the network about the compromise and update the health of the overall network. Often, the core operations at this stage are to compare a string to the most recently encountered strings, as well as list of known virus elements. String matching should not be relied on for future applications, since encryption will not allow string matching to be performed and virus signature will become dynamic. Instead, a system based on a solid learning background and anomaly detection should be considered.

### **3.6 Summary**

This chapter described how hardware solutions are used to prevent virus attacks by string matching methods. While these methods are sufficient for now with static virus signatures, when viruses become dynamic in nature, the methods will not be reliable. That is why the architectural elements of an anomaly detection system were also described in this chapter. We also listed a number of measurable indicators. These behavioral elements of an attack resilient system would allow users of a network to continue normal everyday tasks, while only disrupting the nodes that are violating normal protocol. By normal protocol, it is meant a user's normal traffic pattern, whether that is



only through computer usage or even accessing other areas of their organization. With roaming profiles, these usage statistics can be loaded into the programmable hardware at each station and still monitor that particular user. When an infection does occur though, one cannot only rely on the hardware alone. Predicting where a virus will spread to next will also help to quarantine users and reduce the spread of the malicious attack. To be able to predict where a virus may spread to next though, we turn to known studies on heat-flow/constraint propagation techniques to aid the approach.

## **IV. Label Propagation and Network Labeling**

### **4.1 Chapter Overview**

As explained in the foregoing chapters, a most desirable feature of a high-performance attack resilient system would have the following:

1. A partitioned function between the OS (the item being protected) and the protecting mechanism.
2. The protection mechanism should be programmable, i.e. updatable, and operate at real-time speeds.
3. Capable of detecting insidious anomalies as a routine mechanism.
4. Use collaborative measures to determine if anomalies are wide spread.

Such determinations would require fusing localized measurements from one or more nodes of interest; hence, some incremental computations would be necessary. Then it must be emphasized, communication between cooperative sharing of localized measurements should be separated from normal network traffic. Underlying required communication must take into account the possibility of a corrupt node on the path (i.e. gateway). It is not unreasonable to communicate wirelessly or by other measures. With these given features, we are motivated to develop a framework of iterative estimations of well-being and normalcy using localized measurements from a limited neighborhood.

This chapter expands some basic tools using relaxation techniques, which have been established in computer vision for edge labeling. Using the abstract idea of label propagation, the rules created to label and propagate the node statuses through a computer network are explained. Since this research focused on developing the anomaly detection scheme and how the system would behave, creating these systems is left for

future research topics. Additional topics needed to complete the anomaly detection system are discussed in this chapter and possible solutions introduced.

## **4.2 Relaxation Synopsis**

Relaxation has been used in image analysis to extrapolate parts of an image that are either unknown or have a weak standing compared to other neighboring points. Computer vision techniques improve edge operator measurements by adjusting them based on measurements of neighboring edges [2]. This method can also be applied towards monitoring and updating the security status of a computer network. To apply the technique, local conditions must be observed everywhere within the network and then each node must be classified based on its status. By then applying a compatibility matrix, the confidence of the current status will be updated. Through several iterations, the network will converge to a globally consistent status. To gain an understanding of how a global status can be achieved in computer networks, one must understand the relaxation technique applied in computer vision first.

### **4.2.1 Image Analysis**

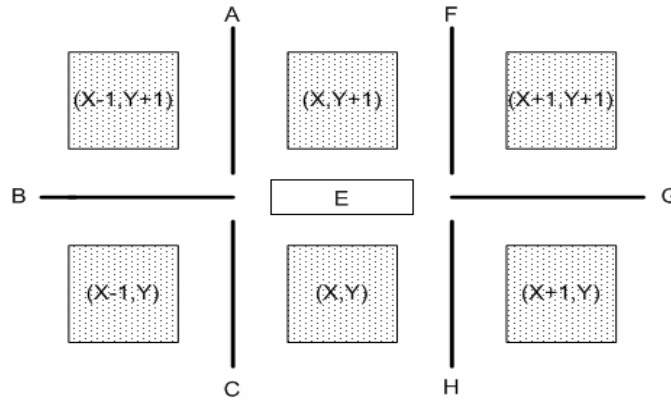
The relaxation technique interactively segments an image by making fuzzy or probabilistic classification “decisions” in parallel. Adjustments are made in successive iterations based on the decisions made in the preceding iteration at neighboring points [27]. Parallel processing improves the processing speed since the relaxation approach is order independent and typically, only a few iterations are necessary. The approach is to recognize local edge patterns, which cause the confidence in an edge to be modified.

There are three possibilities [26]: confidence of an edge can be increased, decreased, or unchanged. An edge relaxation algorithm is in Figure 7. Important parts of the algorithm are step 2, computing the nature of local support or concurrence, and step 3, modifying edge confidence.

0. Compute the initial confidence of each initial edge,  $C^0(e)$ , as the normalized gradient magnitude normalized by the maximum gradient magnitude in the image.
1.  $K=1$ ;
2. Compare each edge type based on the confidence of edge neighbors;
3. Modify the confidence of each edge  $C^k(e)$  based on its edge type and its previous confidence  $C^{k-1}(e)$ ;
4. Test the  $C^k(e)$ 's to see if they have all converged to either 0 or 1. If so, stop; else, increment  $k$  and go to 2

**Figure 7: Edge Relaxation Algorithm [2]**

Edge classification relies on the notation for edges in Figure 8. The edge type is a concatenation of the left and right vertex types, where vertex types are computed from the strength of edges emanating from the vertex [2].



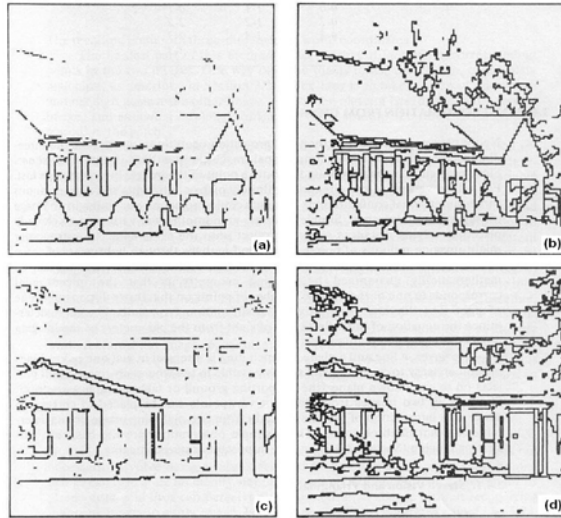
**Figure 8: Edge Notation [2]**

*Each shaded area represents a pixel, whose coordinates are listed. The dark lines represent a pair-wise difference between the pixels they separate. In this snapshot, we seek to estimate the collective support (lack thereof) for the segment E, whose present value is to be updated..*

The local support provided to each piecewise measurement can be analyzed into one of four possibilities in each end. There are sixteen distinct cases, arranged in a matrix form in Table 3. The end result of segmentation seeks a contour. In other words, if an element like “E” were to be in the final result, it must be accompanied by an element at each end. A closer examination of contours in a 4-connected geometry, as the one shown here, reveals that each segment will be supported by exactly one segment at each end in most cases. Also, in some rare cases, one end may have more (one-to-many or many-to-one) i.e.  $1-*$  or  $*-1$ , where  $*(2,3)$ . Examples of the relaxation technique applied to images are shown in Figure 9 [2]. The images on the left, have edges with normalized magnitudes greater than 0.25. Images on the right show the results after five iterations. Weak edges have many gaps in the boundaries for the images on the left, but the confidences of the weak edges are increased, due to neighboring edges and the rules in Table 3.

**Table 3: Edge Type Rules**

		Left Neighborhood Support			
		0	1	2	3
Right Neighborhood Support	0	-	0	-	-
	1	0	+	+	0
	2	-	+	0	-
	3	-	0	-	-
0 : represents no change in confidence “+” : represents an increment “-” represents a decrement					



**Figure 9: Edge relaxation example images [2]**

*Images (a) and (c) are original raw edge data, where images (b) and (d) are results after five iterations of relaxation. Using supporting neighbors with higher confidence in their labeling provided support for the weaker labeled pixels. These weaker pixels eventually gained strength to continue a line in the image, or provide a gap between edges. This type of neighboring support is also needed in computer networks. When a node is unsure of its own state, it can turn to neighboring nodes, with higher confidence, and determine what label fits best to support the surrounding neighbors.*

#### **4.2.2 Network Analysis**

Applying the relaxation technique to network breach indicators, using pair-wise security/trust policies in effect, allows a node to determine if it is likely to be infected and need to be quarantined. If several nodes in a subnet determine they are infected, while the neighboring nodes do not know their status, then using relaxation these “unknown” nodes will adjust themselves toward the infected state. Likewise, if several subnets in a network have been marked as “infected,” neighboring subnets that are either “unknown” or have a “weak secure state” will adjust their status towards the non-secure/infected state. This global dynamic adjustment of status allows a network or its subnets to converge to an overall security level.

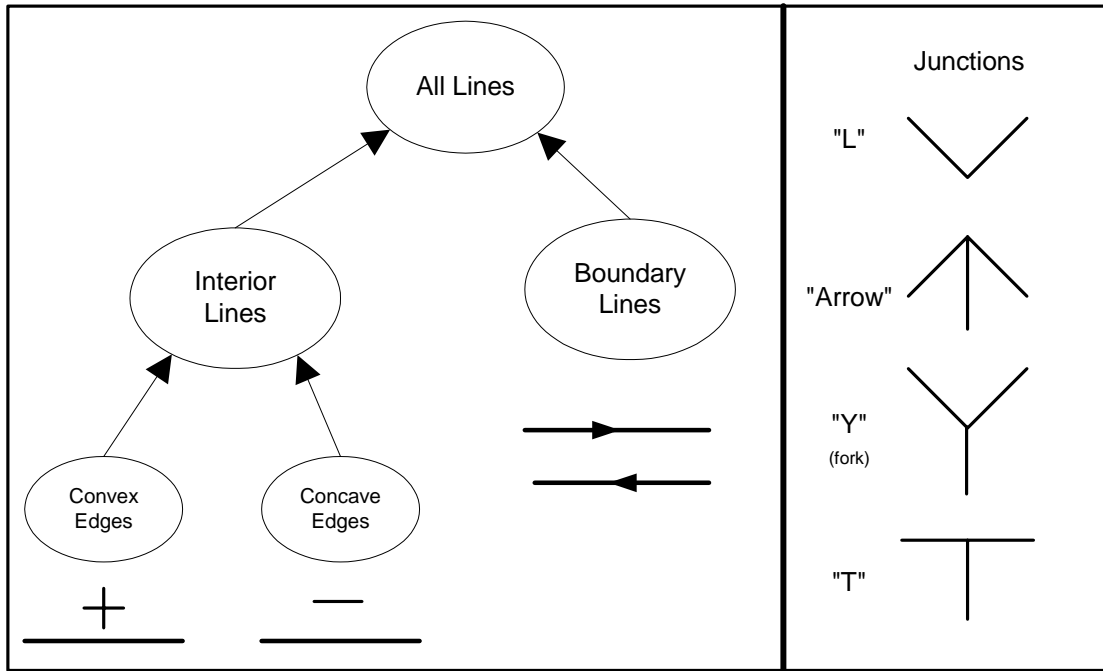
Currently network users protect their systems with anti-virus products and personal firewalls that detect infections or intrusions on only their system. Security products notify the user of the machine, and allow them to take manual or automatic action. Once the program has removed the threat, the user is notified. While this is occurring on one machine, the same threat may be attacking several more machines on the same network. Each attack tries to compromise an individual system. If all users on the network are running the same anti-virus programs (as in domain controlled network) the attack could be prevented more efficiently if once a system detected the threat, it notified other systems to be on alert for an attack, or even notified a centralized gateway of the attack so system wide resources could be used. Creating a distributed detection and notification method allows other users on the network to be aware of potential infection. Current software does not use this technique, which could prevent large outbreaks of viruses and even worms within a network. For software to use distributed threat notification, each platform would need to run compatible software. The heterogeneous nature of these systems makes it a challenge. Within each platform, there may be multiple operating systems (e.g., Windows 3.1, 95, 98, 2000, XP, and Vista) which a single program will not run on. To take advantage of a distributed detection and alert system for all systems on a network including legacy systems, a hardware solution that is operating system and platform independent is a more desirable option. Before this hardware system can be designed though, certain labels and methods of label propagation must be defined beforehand.

### 4.3 Node Labeling and Propagation

We now introduce the global analysis of localized information using a second example in vision. The input is a line diagram, such as one shown in Figure 12. A distinct label to each line segment, as to how it is interpreted, represents the final semantic interpretation. There may be more than one interpretation of a scene, but the propagation of these labels, using rules, helps neighboring junctions to acquire labels consistent with the rest of the image. A similar principle applies to computer networks, labeling a node and propagating the status to neighboring nodes, keeps the overall status of the network current. Developing rules for label assignments is necessary to limit the possible ways a scene can be portrayed and to arrive at a valid interpretation.

Before a flat image can be interpreted into a three dimensional image, the edges of the image must first be analyzed to determine the orientation in a plane. A trihedral block world refers to the context where only three surfaces can meet to create a vertex; and, it includes simple objects such as cubes, boxes, tetrahedrons, do-decahedrons etc. Limiting the object to a trihedral polyhedron, without shadows or cracks, leads to only four line labels [38]. All lines are either a boundary line, with the object on either the left or right side of the line, or an interior line, which can be concave or convex. Without including shadows and cracks, the line diagram of such scenes are fully characterized by junctions and corners whose configurations are limited to one of four types, namely an 'L', 'Y', 'T', and an 'Arrow'. Each of these can manifest in one of several distinct configurations [38]. Figure 10 shows the possible line labels and junction types for trihedral polyhedron referenced in this research.







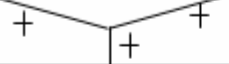


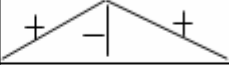
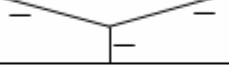
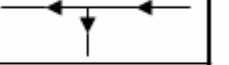
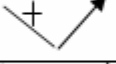
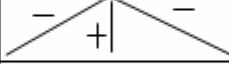


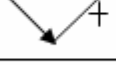
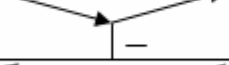
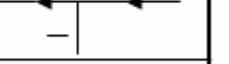
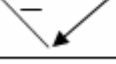


**Figure 10: Line Labels and Junction labels for trihedral polyhedron**

*Lines in computer vision can be split into two separate categories, interior lines and boundary lines. Within the interior lines category, a line can be broken down into a convex edge or a concave edge. Depending on the perspective of an image, a concave line will always protrude towards the viewer. From these line types, four unique junctions can be created. Not all line labels can be used with each junction, so rules will be created further in the chapter to limit the number of junctions an image can possess.*

Although a 'Y' shaped corner made of three line segments, can be labeled in 64 ways, there exist only three valid configurations of a 'Y' junction [38]. An 'L' junction will never be adjacent to a 'Y' junction. A line joining an 'L' and 'Arrow' must be an occluding edge (i.e. two specific labels among four possible labels a line can be assigned with). Using these junctions with the given labels results in an upper bound of 208 possible junction labels. Each line can have four possible labels, where an 'L' junction can have  $4^2 = 16$  labels, and an 'Arrow', 'Y', and 'T' each can have  $4^3 = 64$  ways to label them. Only 18 of these combinations are physically possible though given the assumed limitations [38]. The direction of the arrows for the line labels represent which

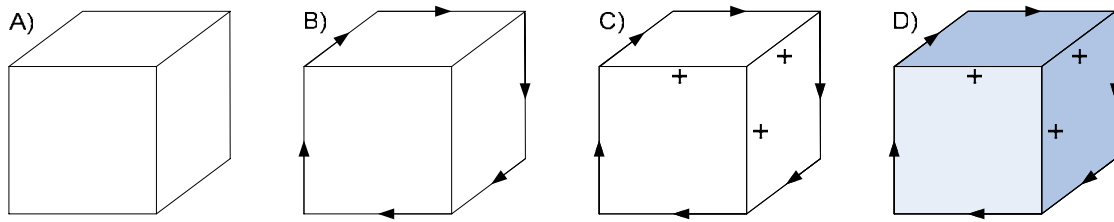
side the object resides. Facing in the direction of the arrow, the object is always considered to be on the right side and the background, or open space, is on the left side. Table 4 shows all the possible junction labels for a trihedral polyhedron with the limiting factors. These labels will be applied to the two dimensional image to help determine a valid three-dimensional shape in space.

**Table 4: All 18 possible junctions for trihedral polyhedron**

L Junctions	Arrow Junctions	Y Junctions	T Junctions
			
			
			
			
			
			

From the table of all possible junctions, one may notice that there is only one possible labeling with a “+” in a Y junction. There is also only one junction for an Arrow with an outside edge “→”. Images containing these junctions benefit from the decreased time it takes to analyze and consistently label the image. Figure 11 is an example of how the labeling of a cube in space can be performed. First (part B), the border lines are labeled first, so the interior lines can be isolated. Next (part C), the Arrow junction’s shafts must be labeled as a “+,” since there are no other possible junctions for an Arrow with boundary lines, which were constrained by the first step. Finally (part D), the Y junction

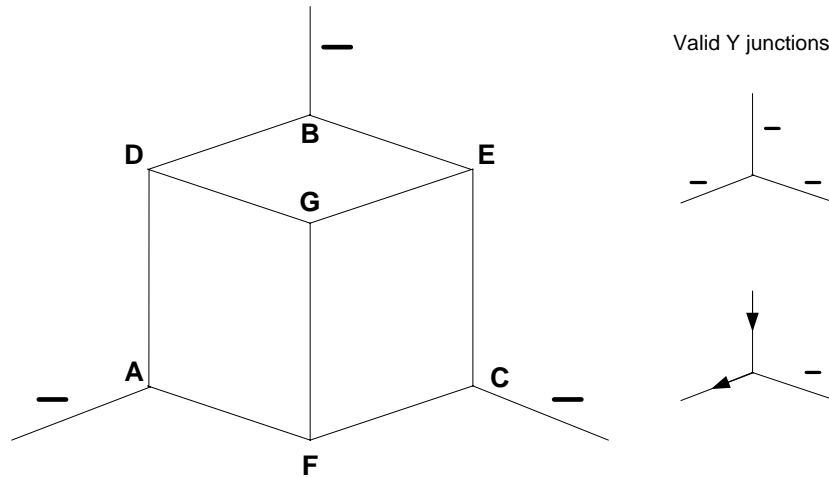
edges have already been labeled, but a check is still needed to make sure it is a valid label for a Y junction. After this check, the object can be identified as a cube with the lightly colored side extending out of the paper the most. Whereas in part A, the object could be seen as either a cube coming out of the page, or as a cube shaped hole cut into the page.



**Figure 11: Consistent Labeling Example**

*A) Original image, B) boundary lines applied, C) propagation of initial labeling conditions, D) final check and object analysis*

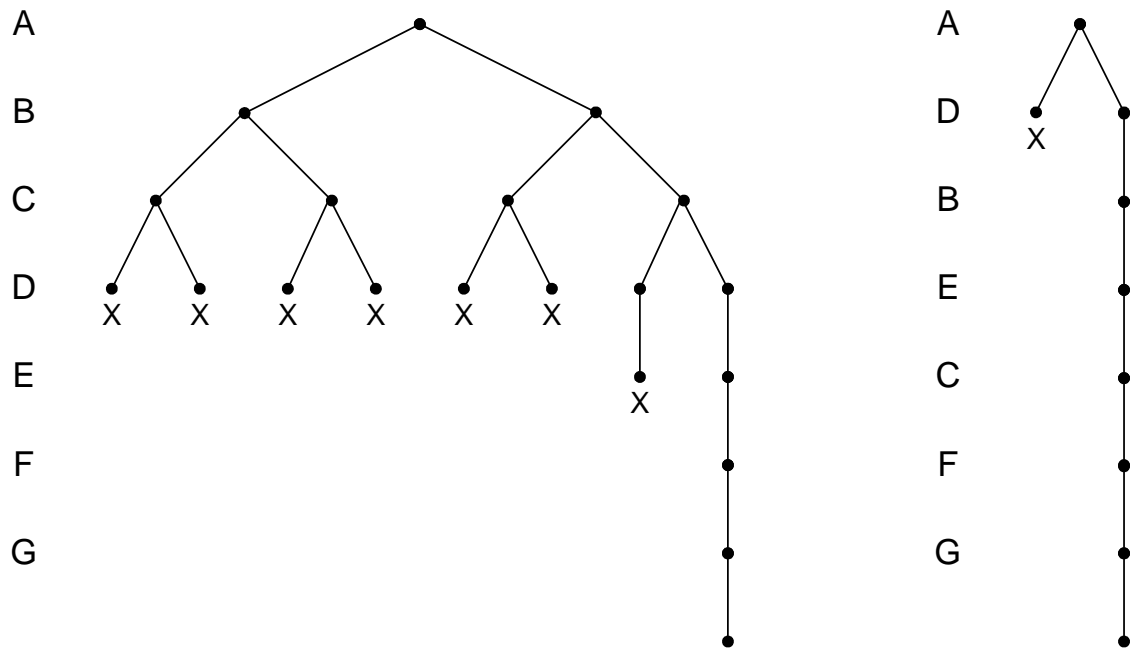
If an object is not floating in space and is in front of a background, then labeling all the boundary edges first may not be possible. In this case, one label may allow the analysis to finish, while another label will fail to create legally labeled junctions. Depending on the order in which junctions are analyzed, the tree of alternatives needed to be searched can change shape. A specific order can force a definite conclusion every time, but this does not always occur [6]. With the initially labeled image in Figure 12, junctions A, B, and C have one edge labeled already. There are two valid Y junctions for this configuration, all “-” edges or 2 boundary edges and a “-” edge. Both junctions must be considered towards a valid solution, but the order in which the junctions are analyzed can mean the difference between a two-branch tree and a multiple branch tree, as shown in Figure 13.



**Figure 12: Two possible labels for the Y junctions of the cube [6]**

*There are two possible labels for the Y junctions in this cube example that are valid. Since neither of these junction labels are forced by neighboring junctions to converge on a single label, a search tree is required to follow all possible valid labels.*

For one search tree, the junctions A, B, and C are first analyzed since each already has one edge labeled, so the search tree will look like the first tree in Figure 13, with junctions analyzed in lexicographic order. This tree branches out rapidly since each of the initial junctions have two possible solutions. If the order were changed, so that after analyzing junction A, the next neighboring junction D is labeled, then a simpler tree is created. By working around the perimeter of the cube with each sequential neighbor, the number of possible choices at each junction is reduced. In any size drawing with any labeling method, the convention has always been to number the boundary junctions consecutively and visit these junctions first [6]. This exploits the extra constraints available from them.



**Figure 13: Depending on the order of junction analysis, a complicated or simple tree is created to search for a valid solution.**

The Waltz Algorithm [38] is another rapid labeling method that iterates toward a globally compatible label. The method keeps a list of possible labels at each node, depending on neighboring nodes. After the first junction has a list of valid labels, the next neighboring junction has a list created for it, but excludes labels that are not compatible with any of the previous junction's labels. In turn, the new neighbor also places additional constraints on the original junction. Continuing this method to other nodes will eventually propagate and reduce the possible label lists for each junction. The goal is to converge on a junction with only one possible label, which will then propagate back through the junctions to converge on a single labeling for each junction. Analyzing an image with this method grows in a linear proportion with the number of lines in the

drawing. Most junctions have only one unique labeling, after only two or three visits in large drawings.

In a general case though, relaxation labeling is an iterative procedure, with the goal of reducing labeling ambiguities and arriving at a global consistency. With an initial labeling assignment, updates are performed based on a compatibility model. Information is propagated, due to the iterative nature of the relaxation process, until all object's labels are consistent or vary only slightly from its previous state. This complex task of global labeling is accomplished by simple, local computations, which is the attractive feature of relaxation. Although a single machine can accomplish this task, parallel architectures can take advantage of these individual local computations, and then converge on a global labeling much more rapidly.

The standard process of relaxation labeling [25] is based on a set of objects  $\mathbf{B} = \{b_1, \dots, b_n\}$ , and a set of labels  $\Lambda = \{1, \dots, m\}$ . Each object of  $\mathbf{B}$  will be labeled with exactly one label of  $\Lambda$ . Using local measurement, a vector  $p_i^{(0)} = (p_{i1}^{(0)}, \dots, p_{im}^{(0)})^T$  is derived for each object  $b_i$  such that  $0 \leq p_{i\lambda}^{(0)} \leq 1$  for  $i=1 \dots n$ , and  $\lambda=1 \dots m$ , and  $\sum_{\lambda} p_{i\lambda}^{(0)} = 1$ , for  $i=1 \dots n$ . Each  $p_i^{(0)}$  is the former probability distribution of labels for the object  $b_i$ . To obtain an initially weighted labeling assignment for the objects of  $\mathbf{B}$ ,  $p_1^{(0)}, p_2^{(0)}, \dots, p_n^{(0)}$  can be concatenated and denoted  $p^{(0)} \in R^{nm}$ . The labels are also assumed to not be randomly chosen, but to have constraints expressed in the terms of an  $n \times n$  matrix  $R$  [25].

$$R = \begin{bmatrix} R_{11} & \dots & R_{1n} \\ \vdots & \ddots & \vdots \\ R_{n1} & \dots & R_{nn} \end{bmatrix}$$

Each  $R_{ij}$  is an  $m \times m$  matrix of non-negative real-valued compatibility coefficients. The strengths of compatibilities between  $\lambda$  on object  $b_i$  and  $\mu$  on object  $b_j$  is measured by the coefficient  $r_{ij}(\lambda, \mu)$ . Higher values represent better compatibility, while lower values represent more incompatibility between the objects [25].

$$R_{ij} = \begin{bmatrix} r_{ij}(1,1) & \dots & r_{ij}(1,m) \\ \vdots & \ddots & \vdots \\ r_{ij}(m,1) & \dots & r_{ij}(m,m) \end{bmatrix}$$

Combining the constraints matrix with the compatibility matrix, results in an  $(n*m) \times (n*m)$  matrix for a graph with  $n$  nodes and  $m$  unique labels.

$$R = \begin{bmatrix} \begin{bmatrix} r_{ij}(1,1) & \dots & r_{ij}(1,m) \\ \vdots & \ddots & \vdots \\ r_{ij}(m,1) & \dots & r_{ij}(m,m) \end{bmatrix}_{(1,1)} & \dots & \begin{bmatrix} r_{ij}(1,1) & \dots & r_{ij}(1,m) \\ \vdots & \ddots & \vdots \\ r_{ij}(m,1) & \dots & r_{ij}(m,m) \end{bmatrix}_{(1,n)} \\ \vdots & \ddots & \vdots \\ \begin{bmatrix} r_{ij}(1,1) & \dots & r_{ij}(1,m) \\ \vdots & \ddots & \vdots \\ r_{ij}(m,1) & \dots & r_{ij}(m,m) \end{bmatrix}_{(n,1)} & \dots & \begin{bmatrix} r_{ij}(1,1) & \dots & r_{ij}(1,m) \\ \vdots & \ddots & \vdots \\ r_{ij}(m,1) & \dots & r_{ij}(m,m) \end{bmatrix}_{(n,n)} \end{bmatrix}$$

An initial input labeling to the relaxation algorithm would be appear in the form of  $p_i^{(0)} = (p_1^{(0)T}, \dots, p_n^{(0)T})^T$ . This would be updated iteratively, according to the constraint

model, so global consistency would be achieved. At step  $t$ , updating labels are performed by the following formula [25]:

$$p_{i\lambda}^{(t+1)} = \frac{p_{i\lambda}^{(t)} q_{i\lambda}^{(t)}}{\sum_{\mu=1}^m p_{i\mu}^{(t)} q_{i\mu}^{(t)}},$$

where the denominator is simply a normalized factor, and the strength of support that is given to  $\lambda$  for being the correct label for  $b_i$ , at step  $t$  is represented by:

$$q_{i\mu}^{(t)} = \sum_{j=1}^n \sum_{\mu=1}^m r_{ij}(\lambda, \mu) p_{j\mu}^{(t)}$$

Ideally, this process will iterate until the values do not change anymore, meaning a steady state has been reached. In reality though, there will still be slight continuous variation in the values, or label probabilities, so a minimum difference would be set to provide a stopping point. Also, in practical applications, the numbers of objects that interact are usually limited. This is due to the fact that objects usually only interact in a small neighborhood, therefore reducing the matrix size, and the computational time needed to label all the objects. Interacting within a small group of neighbors is also inherent with computer networks, so similar principles can also be applied.

#### 4.4 Canonical Node Types

Using the ideas from line labeling, first a set of interactive nodes are needed, then meaningful labels regarding their status are needed and finally a set of rules to follow must be created. This section will setup a small interactive network and show how the

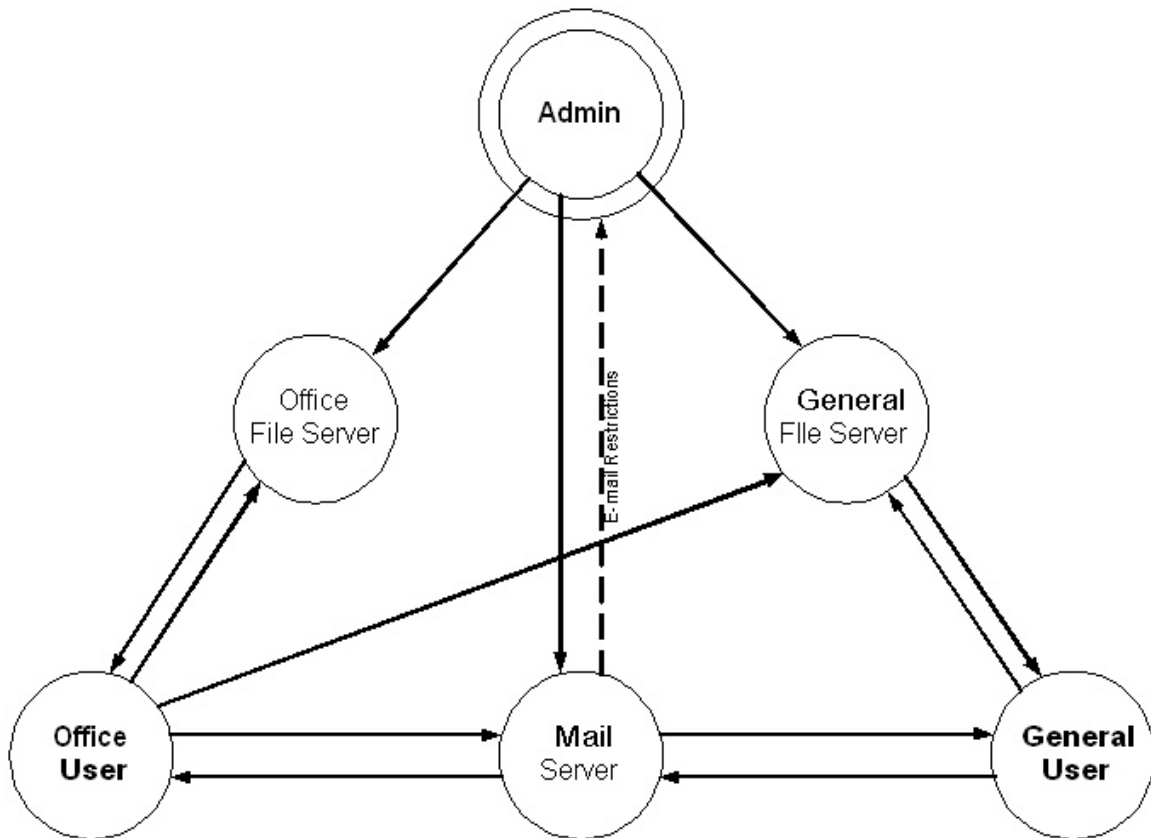


relaxation labeling methods will apply to network node labeling and virus propagation prediction.

A typical university office network is considered for example. It has are three types of computer users: administrators, office oriented users, and general users. Each of these users interacts with each other in the network in the following ways: All users have access to the mail server and they can be infected by an attachment, except administrators, who have a policy to never open an attachment from non-administrators without a thorough virus scan. There is also a general file server, where office users can dump information and files, but do not execute programs or files from this location. Office users have their own file server, where they can transfer and execute programs from other office users. General users have access to the general file server and can read and write data to this server. These general users can be infected by other general users, office users, or even administrators, through e-mail or file server usage. Office users can be infected by other office users, general users (only by e-mail), and administrators. Other administrators can only infect administrators. In this situation though, a member of the administrator must bring an infected file from outside the network, to start the spread of infection within the administrator group.

Figure 14 shows the network structure and communication tree. In this scenario, it is also assumed that the actual file servers themselves cannot be infected, but only the files they host. Also, it is assumed that only a single event occurs during each time interval. In other words, only one email is sent, or one user accesses a single file. This

allows one to follow through a simple sequential labeling example. The actual process would allow multiple events to occur in parallel, but following an example is difficult.



**Figure 14: Network layout and user interaction for simulated scenarios**

Creating rules from the given assumptions and description would involve two nodes. The method of how an infection can be passed from one to the other is subject to the local pair-wise security/trust in effect. Table 5 illustrates, that there are seven ways that an infected email can be passed from one source to another. In addition, in the same table, five ways an infected file can be transmitted are shown. For actually propagating the virus in a network, it is essential to have a three-node infection label or two edges.

**Table 5: Two-node virus propagation from e-mail and file execution**

**Infections by E-mail**

		Neighbor at risk		
		Admin	Office User	General User
<b>Infected User</b>	Admin	+	+	+
	Office user	0	+	+
	General User	0	+	+

**Infections by File Execution**

		Neighbor at risk		
		Admin	Office User	General User
<b>Infected User</b>	Admin	+	+	+
	Office user	0	0	+
	General User	0	0	+

*In these tables, “+” denotes that it is possible for the infected user to propagate a virus to the user being infected. A “0” represents that it is not possible to transmit the virus, either because of rules applied to the network, or the physical connections are not available.*

The infection comes from one node and is transmitted to another. This newly infected node can then pass the virus on to another non-infected node. Graph edges will represent the way the infection is spread. In this example, edge infection types can be labeled two ways. Infection transmitted by e-mail or by file transfer and execution through a file server. Since there are two methods of infection, and a two edge labeling scheme, there are four different ways a three node communication can be labeled. The original node can send a virus by e-mail to the middle node, which passes on the virus by e-mail to infect another node. In scenario two, an infected user deposits a file to a server and another user accesses the file to become infected. Then that user either leaves the file, or replaces it with another infected file, which then infects a third user. The third and fourth

scenarios combine propagation effects. Originally, the virus is transmitted by an e-mail, so recipient of the e-mail becomes infected and then accesses a file server and infects a file. This propagates the virus to a third user who accesses the file. The final scenario is the opposite of the third. Instead of the virus propagating by e-mail first, an infected user accesses a file server and infects a file that is soon accessed by a second user. This newly infected user then sends an e-mail which includes the virus to another user. Table 6 shows all the possible labels with three interacting nodes.

A graph of the virus propagation, using the newly formed labels, could result in the entire network becoming infected if a weighting system is not put on the edges between users. Creating a compatibility matrix of communication methods from a user, would reduce the possibilities where a virus could spread. To further limit the possible virus propagation, a more detailed compatibility matrix would be created between each user, calculating the usage of e-mail as a communication means, or the type and number of files accessed by the user, which have also been accessed by other users. From this, using iterative techniques, labeling which nodes are possibly infected and where the virus has spread to can be calculated. In addition, the path of the virus can also be predicted from nodes that have not been affected, by use of their individual, yet weighted, compatibility matrices. Once one node is identified, without a doubt, as being infected, the path from where the virus originated could also be predicted, using backtracking. Since the compatibility matrix represents the normal actions from a node, anything abnormal could be detected and help aid in the restructuring of the prediction path.

**Table 6: Propagation paths**

*a) A virus is passed by e-mail and then again by e-mail, b) An infected file is deposited, then accessed and infects another user, who deposits another infected file which is accessed, c) The virus is passed by e-mail originally and then an infected file is deposited to the server that a user accesses, d) A virus is contained in a file placed on the server, which is then accessed, and that infected user passes the virus via e-mail.*

**a)**

<b>E-mail to E-mail Propagation</b>		
Origin	Middleman	Infected
Admin	Admin	Admin
Admin	Admin	Office User
Admin	Admin	General User
Admin	Office User	Office User
Admin	Office User	General User
Admin	General User	Office User
Admin	General User	General User
Office User	Office User	Office User
Office User	Office User	General User
Office User	General User	Office User
Office User	General User	General User
General User	Office User	Office User
General User	Office User	General User
General User	General User	Office User
General User	General User	General User

**c)**

<b>E-mail to File Propagation</b>		
Origin	Middleman	Infected
Admin	Admin	Office User
Admin	Admin	General User
Admin	Office User	Office User
Admin	Office User	General User
Admin	General User	General User
Office User	Office User	Office User
Office User	Office User	General User
Office User	General User	General User
General User	Office User	Office User
General User	Office User	General User
General User	General User	General User

**b)**

<b>File to File Propagation</b>		
Origin	Middleman	Infected
Admin	Office User	Office User
Admin	Office User	General User
Admin	General User	General User
Office User	Office User	Office User
Office User	Office User	General User
Office User	General User	General User
General User	General User	General User

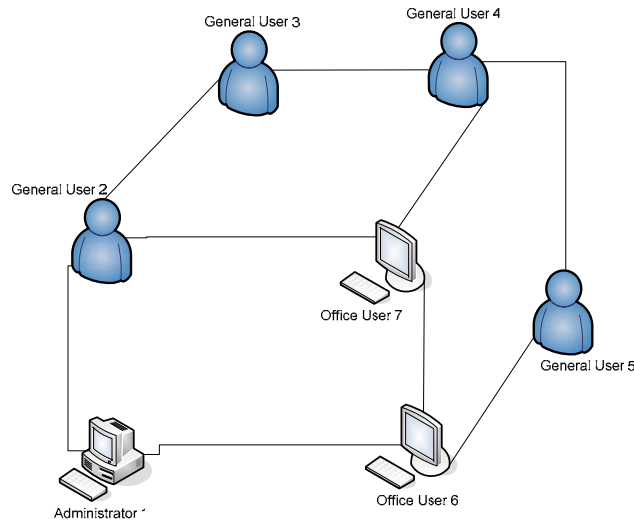
**d)**

<b>File to E-mail Propagation</b>		
Origin	Middleman	Infected
Admin	Office User	Office User
Admin	Office User	General User
Admin	General User	Office User
Admin	General User	General User
Office User	Office User	Office User
Office User	Office User	General User
Office User	General User	Office User
Office User	General User	General User
General User	General User	Office User
General User	General User	General User

The purpose of having at least a three user rule set is that when only two users are used, relaxation labeling can bounce between the two users, if their communication to each other is predominate over other connections. Using the three rule layout prevents the compact loop of predicting continuous infection between only two nodes.

## 4.5 Infection Propagation Scenario

To demonstrate the effectiveness of relaxation in a computer network, a small example will be used. The main compatibility matrix can be created of how users communicate. It can be analyzed in matrix form or converted to a graph. Figure 15 represents the graph form of communications in the simulated network, which for simplicity, represents the Necker cube example. File servers have been removed from the figure to reduce line clutter, although actual communication will still take into account for these file servers. From the figure, one is unsure how these users interact.



**Figure 15: Example network scenario for labeling and relaxation.**

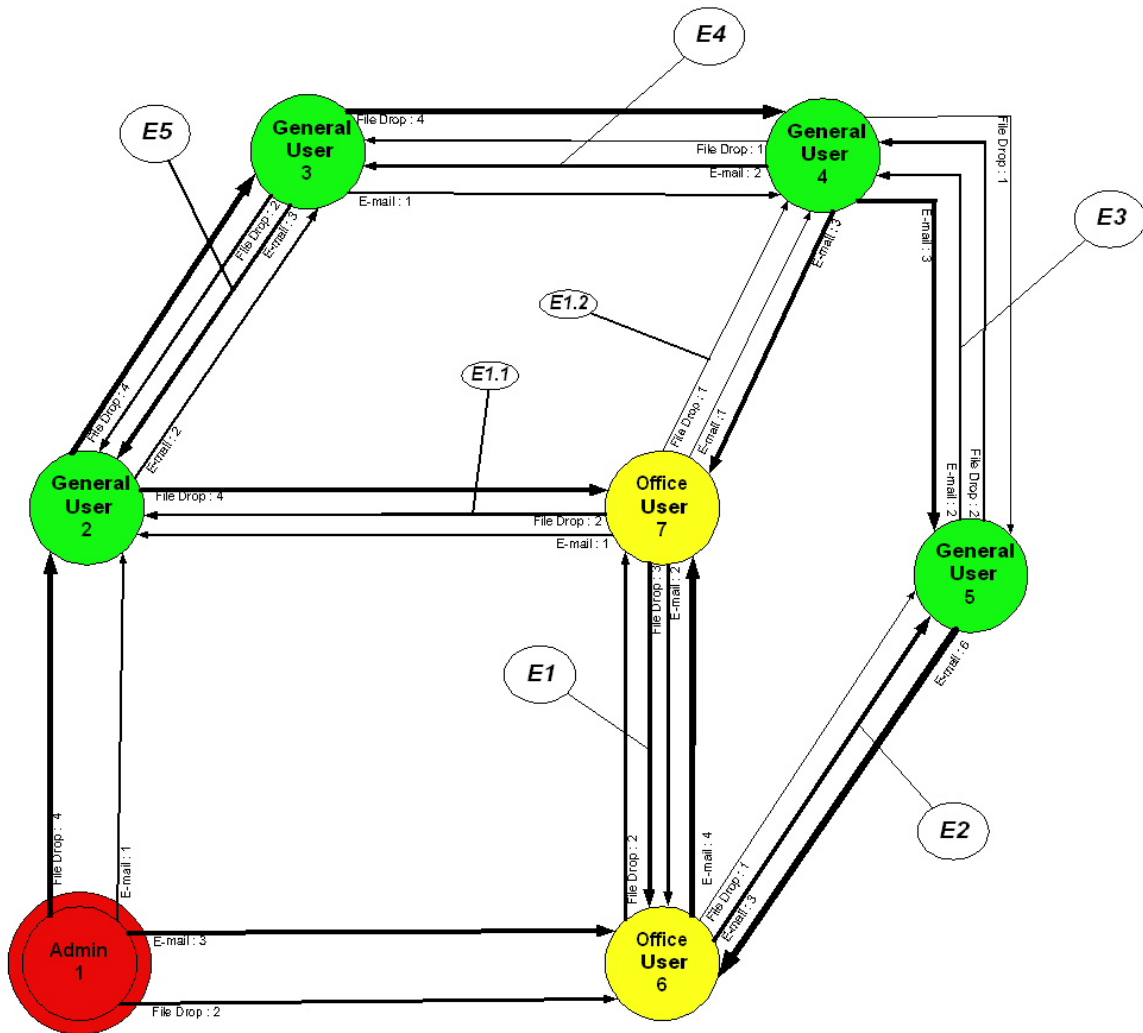
For example, it appears that either *General User 2* or *Office User 6* could infect *Administrator 1*, but that is not the case, when the rules from Table 6 are applied. Even after these rules are applied to the graph, every means of communication, e-mail and file execution, are not performed by each user. To specify further, a compatibility matrix is created to show the common usage of communication means between users. This

compatibility matrix could be created from a combination of user logs and network traffic logs. For this example, a weight system from one through ten is applied to each communication method. Table 7 shows each user's compatibility matrix, whose weights sum to ten, with the highest number representing the most common outgoing communication means to the user. Just as each user has a matrix that shows their most common outgoing interaction method, an incoming interaction method matrix can also be created for each user, from the outgoing compatibility matrix. After defining a network

**Table 7: Compatibilities between users - Larger value represents most used interaction method**

<b>Administrator 1</b>		
<b>Outgoing Interaction with Node:</b>	Sending E-mail	File Drop
General User 2	1	4
Office User 6	3	2
<b>General user 2</b>		
<b>Outgoing Interaction with Node:</b>	Sending E-mail	File Drop
Administrator 1	0	0
General User 3	2	4
Office User 7	4	0
<b>General user 3</b>		
<b>Outgoing Interaction with Node:</b>	Sending E-mail	File Drop
General User 2	3	2
General User 4	1	4
<b>General User 4</b>		
<b>Outgoing Interaction with Node:</b>	Sending E-mail	File Drop
General User 3	2	1
General User 5	3	1
Office User 7	3	0
<b>General User 5</b>		
<b>Outgoing Interaction with Node:</b>	Sending E-mail	File Drop
General User 4	2	2
Office User 6	6	0
<b>Office User 6</b>		
<b>Outgoing Interaction with Node:</b>	Sending E-mail	File Drop
Administrator 1	0	0
General User 5	3	1
Office User 7	4	2
<b>Office User 7</b>		
<b>Outgoing Interaction with Node:</b>	Sending E-mail	File Drop
General User 2	1	2
General User 4	1	1
Office User 6	2	3

topology and compatibilities between nodes, a detailed graph of the network is shown in Figure 16. From this figure, one can visualize the path with the most throughput, where a virus could spread. Larger/bolder edges represent more common interaction modes between nodes. Thinner edges show where a virus may be slow in its spreading capabilities



**Figure 16: Weighted connections, compatibility graph, and virus spreading events for the network example of relaxation methods**



Now that user interaction rules have been defined, as well as a layout and compatibility matrix, assume *Office User 7* has been infected with a virus. Figure 16 also shows the events (*E1*, *E2*,...*E5*) that propagate the virus through the provided network. From the compatibility matrix, the most likely host to be infected next would be *Office User 6*. The method of infection would be through the office file server (*E1*), where *User 7* would place a file, which *User 6* would access and contract the infection. *User 6* then has two possible connections, where the virus could spread. Checking the rules show, that *Administrator 1* is not in danger of becoming infected through *User 6*. The only other user that *User 6* has normal communications with is *General User 5*. The rules show this is a valid connection, for both e-mail and file transfer. The compatibility matrix then predicts that e-mail (*E2*) is the method the virus would most likely spread through to *General User 5*. The only valid user *General User 5* could possibly infect is *General User 4*. From the compatibility matrix though, infection through e-mail and file transfer are equally likely. In this case, two separate propagation cases must be considered. By default, the way a virus is received, will also be the preferred method it is propagated, unless there is a large (greater than two) difference in the compatibility matrix weights. To use this information, the neighboring user(s) of *General User 4* must be analyzed for which is most likely to receive the virus. Since *User 4*'s only choice to propagate is to *General User 3* and the weight of e-mail propagation leaving *General User 3* is higher, infection of *User 4* is predicted to be through e-mail (*E3*). The virus then would be passed on to *General User 3* via e-mail (*E4*), if there were no interaction

to stop the spread of the virus. In the described scenario, after a period of time, all the users, except the administrator would become infected. Other network layouts and examples would not necessarily end in this fashion. Parts of networks would be isolated, due to rules and propagation patterns. Parallel iteration was not fully demonstrated in this example to keep confusion down, but through the relaxation algorithm, three decision trees would have been created from the originally infected *Office User 7*. The virus could have propagated through three possible paths of commonly connected users: *General User 2 (EI.1)*, *General User 4 (EI.2)*, and from the provided example, *Office User 6 (EI)*. In the same sense as predicting where a virus will be propagating, predicting where the virus has been is also possible. Using the compatibility matrix, along with backtracking techniques, a predicted path could be followed to the origin of the virus. Since this method of forensics is “after the fact,” it is not covered in depth by the research.

#### **4.6 Future Modeling and Labeling Issues**

While heat flow and relaxation have been widely studied topics, their techniques have not been applied to computer networks. The extent, at which these proven methods would be effective in a real world network, would need to be measured. The provided scenario in the previous section was an example of a simplified case. Future research would take the network scenario and increase the number communication methods along with adding additional services such as a print server, separate file and application servers, and a web server. Communication to other users over secure connections would increase complexity, but would more accurately model a large-scale network. Including

a Honeypot in the network, would allow a node that ideally would never see traffic or become infected, but still monitor the health of the network. This would essentially be a virus detection sensor, embedded into the network, while still providing valuable information without being a productive system to the users. Infection methods would increase beyond the two provided in this research, infection via macros, java scripts from web pages, and self-propagating worms should be investigated. This increased network model would also increase the size of the compatibility matrix for each user's behavior pattern. Implementing these compatibility matrices into hardware, for each user would allow the processing of this information to be independent of the end-user's systems. These additional research topics would advance the virus prevention methods and decrease the spread in future systems.

#### **4.7 Summary**

As covered in this chapter, we make a compelling introduction for the use of iterative estimations in predicting virus propagation and have suggested a framework of relaxation techniques applied to the network security field. However, we have yet to apply the technique in a large-scale network due to envisioned complexity and a limited timeframe. An overview of relaxation, applied to image analysis and then transferred to network analysis, is provided to familiarize the reader with the computer vision topic.

The rest of the chapter demonstrated and discussed the proposed method of applying line labeling techniques and relaxation to computer networks in order to predict virus propagation. Through the provided example, it was shown that label propagation can be aided with the use of a compatibility matrix at each node. This matrix provides

the information of how connections are made, and which users communicate within the network. Although the background of line labeling and relaxation lie in computer vision, this research has provided parallels between the two topics for these methods to be applied to future network security applications. Along with behavioral aspects a system incorporating these methods should follow, several complex problems that may arise in future work will be explained in the next chapter.

## **V. Future Research and Recommendations**

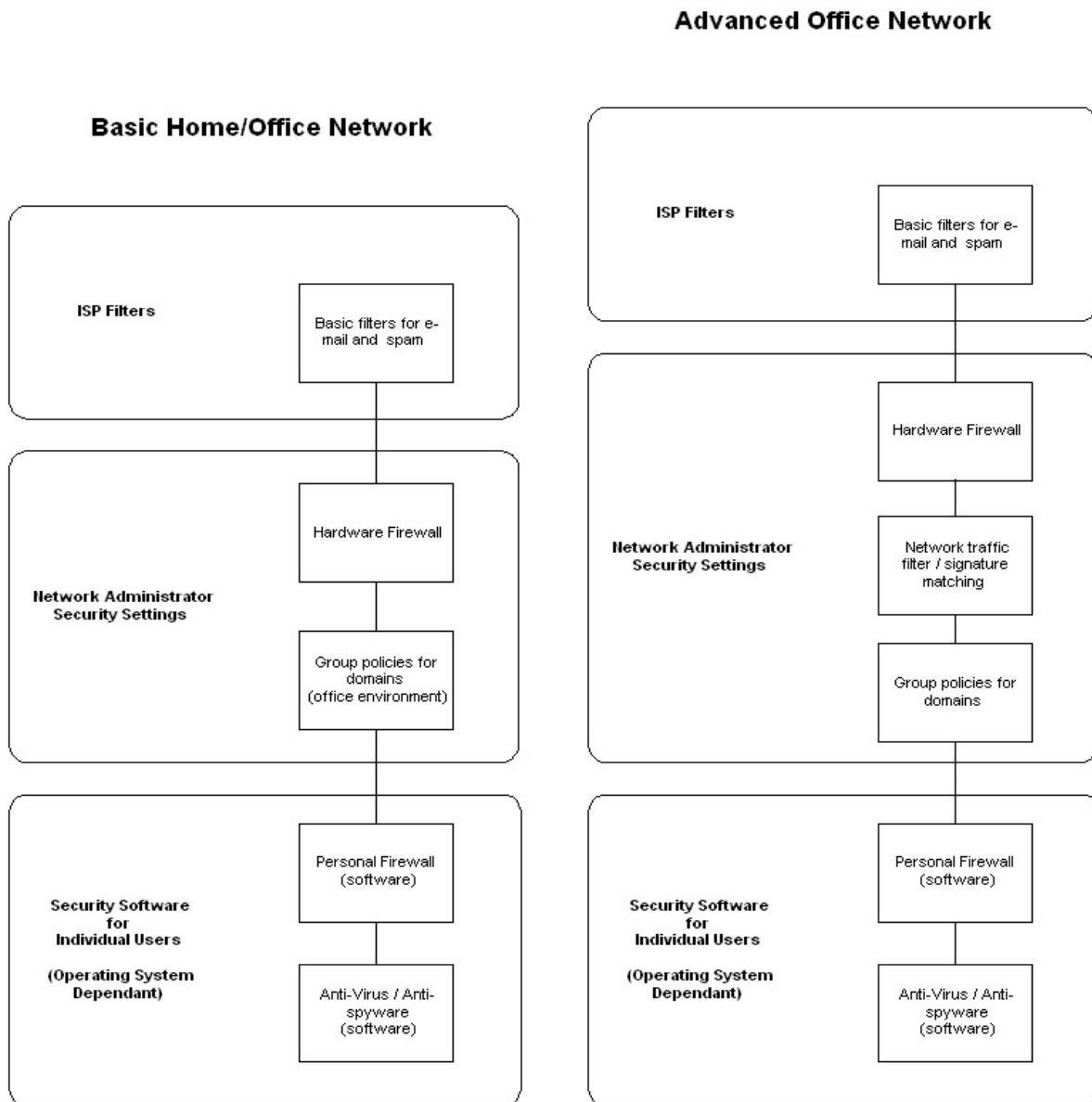
### **5.1 Chapter Overview**

The purpose of this chapter is to show where a hardware solution would fit into a current home or office network. Instead of re-designing a current network, the device should be able to be inserted into the network with little interference to the current setup. This intermediate step is important towards a larger change in network security and analysis techniques. In addition, this chapter provides a basis for future work and describes the theory behind an Anomaly Detection and Prevention Unit (ADPU) and its two-layer small-world network model. Interaction between these devices is explained, so parts of the network can be isolated in case a threat compromises a host on the network. Even though examples of specific system components are described and provided, newer technology may better handle the designed architecture. Limitations of the proposed components are also described.

### **5.2 System Behavior**

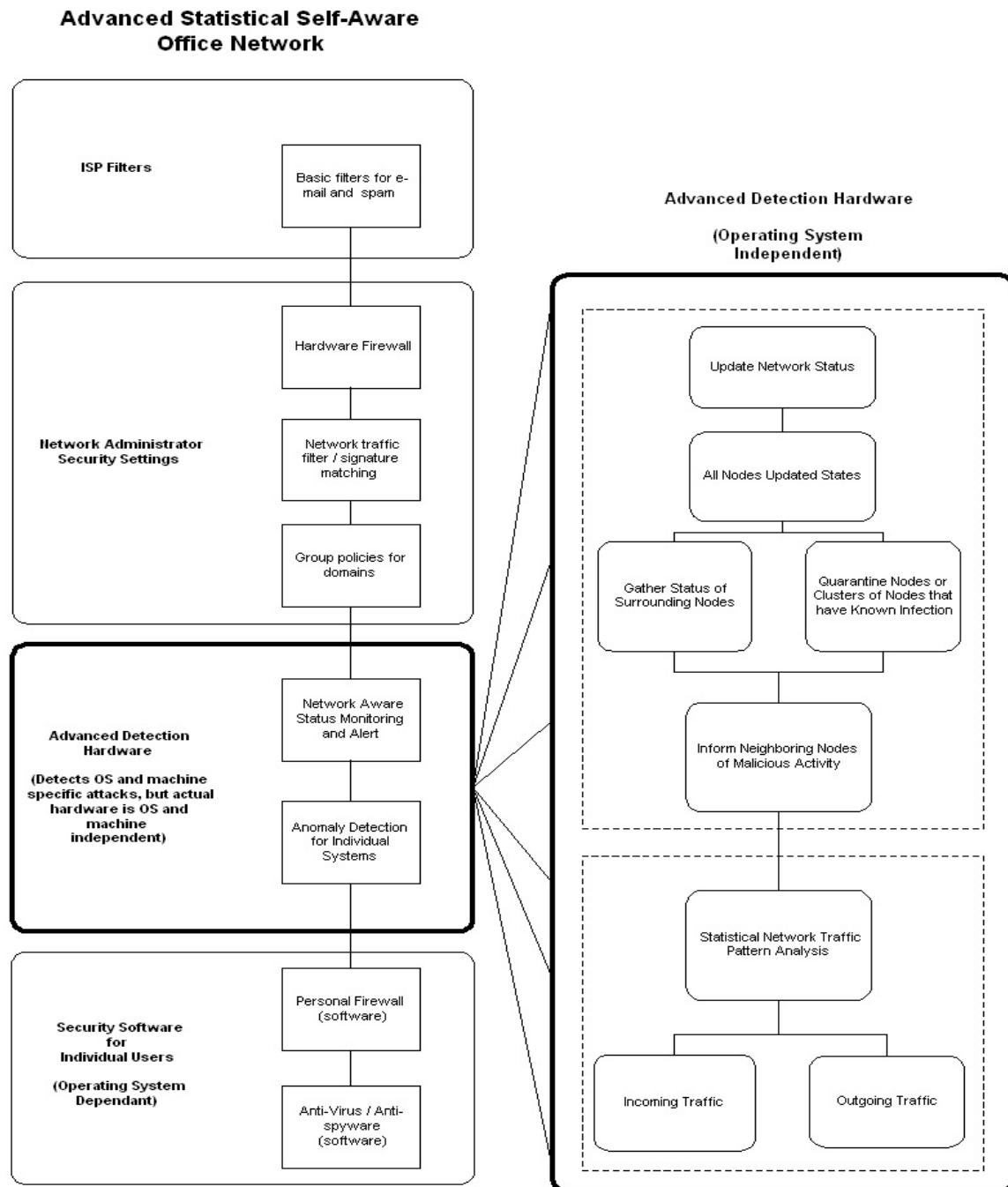
Developing a system that can be used with current network configurations is an important factor for a new device to become accepted in the networking community. To demonstrate how the device would be incorporated into current configurations, Figure 17 shows a typical breakdown of two advanced networks. The top level contains the Internet Service Provider, which has minimal filters for e-mail and spam. Administrative security services make up the second level. These services include hardware firewalls, intrusion detection systems, and group policies for domains. The bottom layer contains machine specific (operating system, type, and version) security that must be implemented

on a per machine basis. Implementing the device, that will be described later in the chapter in more detail, would be incorporated into a standard or advanced network in a block configuration. It would not have to ingrain itself in current hardware, or need major revisions of current network configurations. Figure 18 demonstrates how the advanced hardware detection and prevention system fits into the advanced office network model. In the advanced hardware block, anomaly detection/prevention, network monitoring, and updating are performed. Incoming and outgoing traffic are monitored from a statistical standpoint, any traffic that does not fit into a users “normal” traffic pattern is flagged. This monitoring of traffic is operating system independent, and cannot be easily detected by an outside source. However, the programmability permits machine specific remedies, including source-dependent exceptions. Now that the basic execution strategy has been explained, the behavior model of this device can be covered in more depth.



**Figure 17: Typical network block diagram**

Typical home and office networks contain three basic blocks of security. These include operating system dependant programs such as anti-virus and personal firewalls. The next higher level of security is contained in Administrator settings such as domain settings and the physical setup of the network. This would include group policies and a hardware firewall built into a router. The highest block of security (least user specific) is accomplished by filtering basic content at the Internet Service Provider. The difference between a typical home/office network and a more advanced office network is that in the Network Administrator block, another device may be included to perform signature matching of traffic and filter out possible malicious packets before they can reach an end-user.

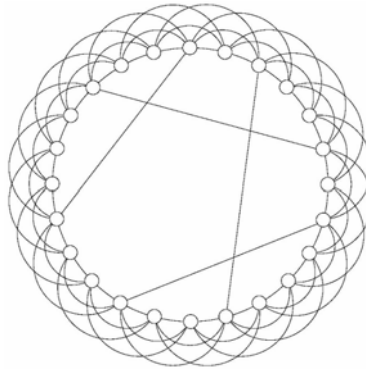


**Figure 18: FPGA implemented network anomaly detection block diagram**

*Inserting advanced detection and prevention hardware is done by simply adding another block to the typical network structure. In this hardware block, monitoring, analysis, detection, and prevention can be performed on each individual end-user. The status of each user would also be determined and neighboring advanced hardware would be notified of possible future malicious traffic.*



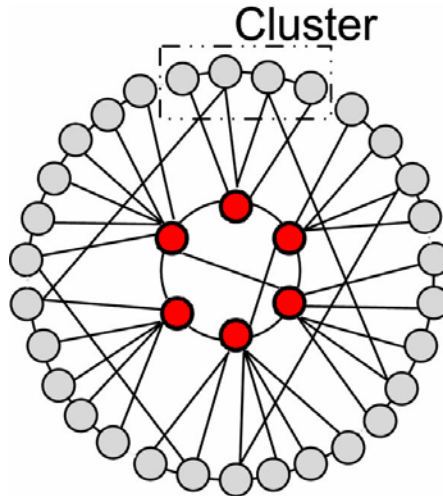
The behavioral model of the system is based on a study of virus and anti-virus dynamics of a network [12]. Since the spreading dynamics of most viruses depend on the underlying network, a representative network model called the small-world network model shown in Figure 19 is used. It is a ring lattice of  $L$  nodes,  $k$  nearest neighbors, and  $S$  shortcuts between random node pairs.



**Figure 19: Single Layer Small-World Network model with  $L=24$ ,  $k=3$ , and  $S=4$  [12]**

This basic model is modified into a two-layer model where the outer level is a cluster and the inner is a Superusers layer. The cluster layer is where virus spreading occurs and the Superuser layer captures the anti-virus process. The cluster layer, which is made up of end-users, receives warnings from the administrators/Superusers and takes precautionary measures as advised by the administrators. This may be done by updating antivirus definitions or closely monitoring e-mail attachments. End-users have bi-directional links between other end-users, but only one-way links coming from the administrators. The administrative layer monitors and detects abnormal events through peers in the cluster layer. If a virus is detected, the nodes in this layer inform their peers (other Superusers). Each Superuser sends warnings to the end-users under their administration (clusters) only. Because of the one-way link to the cluster level, it is assumed that the Superuser

level cannot be infected by a virus. Figure 20 shows how this two-layer model differs from the single layer.

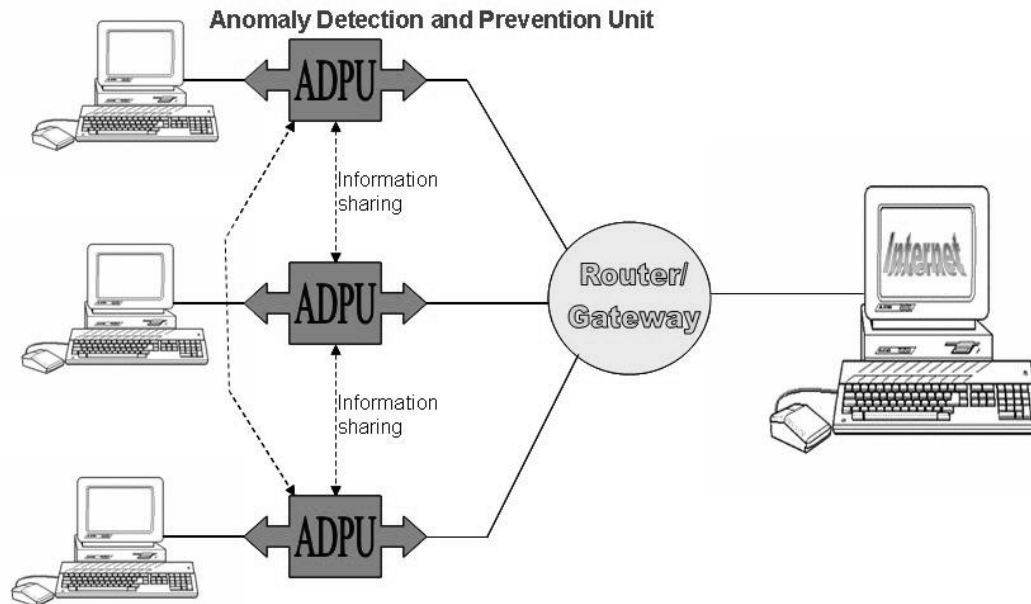


**Figure 20: Two-layer Small-World Network model [12]**

To determine the status of each node under a Superuser, a state machine records the condition of each node. The states an end-user can be in are: Susceptible, Infected, or Immune. Susceptible is the default state for a node when it is introduced into the network and when the network is not seeing any virus activity (steady state). Superusers also have states, even though they cannot be infected. The states of the Superuser level record anti-virus information when an attack has occurred. These states are: Unknown (which is default) or Known. This model of system state is known as the SIM (Susceptible-Infected-iMune) model. Similar models to this one are the SIS (Susceptible-Infected-Susceptible) and the SIR (Susceptible-Infected-Removed) model [12].

The Anomaly Detection and Prevention Unit (ADPU) uses the SIR states for the end-users, so an infected (I-labeled) user can be removed and analyzed after infection.

Figure 21 shows an example of how the ADPU would be implemented in a typical network. The system is placed inline with each end-user's system. Individual units



**Figure 21: Anomaly Detection and Prevention Unit implemented in a small network**

permit fine-grained detection customized for user specific traffic. Another benefit is that a smaller set of behaviors can be learned. The behavior will be more precise for a single user than for a LAN anomaly detection scheme. Most commercial intrusion detection products perform detection at the router/switch level, which reduces cost and locates the hardware in one centralized location to update. A problem with centralized detection is it is a single point of failure. If the switch/router is compromised, the whole LAN is vulnerable. Most IDSs do not monitor traffic within the LAN, only what is coming in from outside the network. A major benefit of the ADPU is the monitoring of incoming and outgoing packets as well as packets within the network among “trusted” systems. Inline host monitoring also prevents a centralized point of failure while still monitoring

traffic. If a single ADPU is compromised, the whole system is not jeopardized and the other units can still prevent systems from being attacked.

Unlike most IDSs, anomaly detection does not use signatures to detect attacks. Therefore, encryption will defeat it. Anomaly detection can still detect where the connection is from, when it was made, and for how long, with or without encryption [31]. In addition, connections can be related to the user's habits throughout certain times of the day.

### **5.3 Auxiliary Support Network**

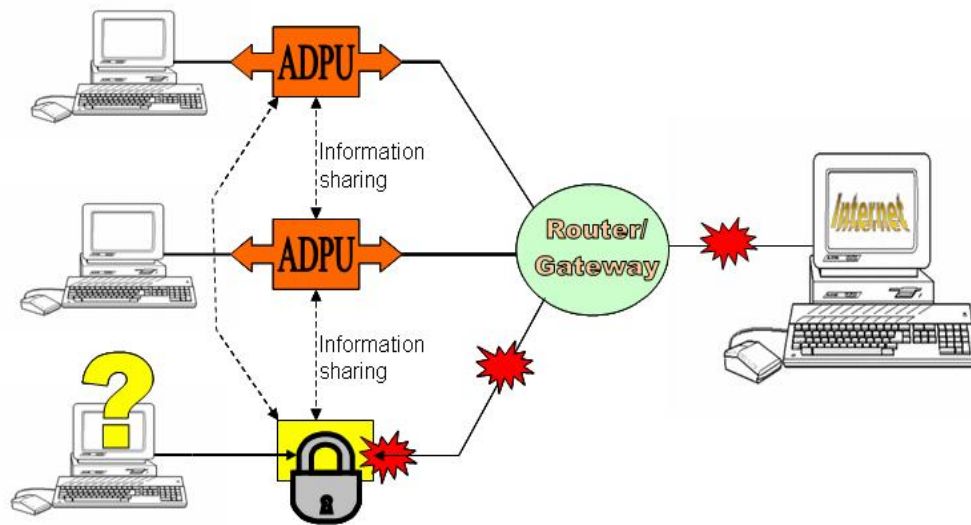
Connections between nodes allow them to communicate their network states. When a device detects abnormal malicious traffic, it can send advanced warning of the attack to its neighboring nodes. Those nodes can transmit to their neighbors until the whole network has been informed of a potential attack. Having regular network status updates, without main-line traffic congestion, is an effective way of relaying information. An additional benefit of communication between these network devices is users can troubleshoot connectivity problems. Rather than having the status of each ADPU relayed through other switches in the primary connection mixed in with data packets, a direct and dedicated connection to a set number of neighbors can communicate the status much quicker. This ad-hoc network approach will preserve clear and usable communication pipeline between security devices even if other network components are disabled. A viable solution for the hardware is a wireless communication path, but certain interfering signals (e.g. cordless phones, wireless video cameras, or jamming devices) could prevent this technology from being used. Other methods could be used such as phone lines,

transmission through power lines, and even a secondary cat-5 network dedicated only for security hardware communication. With any of these methods, there are drawbacks, of adding additional lines, or interference. In any case though, policies must be setup to determine how these devices will communicate and operate.

#### **5.4 System Policies**

System policies can be divided using two scenarios. The flow chart in Appendix 1: Full Flow Chart of ADPU System shows all systems policies. The two scenarios that describe this full system behavior are incoming activity, either from outside the network or an internal threat, and outgoing anomalous activity, which can have a destination outside of the network or to another host in the LAN.

Suppose, as in Figure 22, incoming activity is detected but the ADPU has not determined if the end-user is infected or if it has prevented the attack from completing. The current state of that ADPU is “Unknown,” where the end-user node is now labeled as “Infected.”

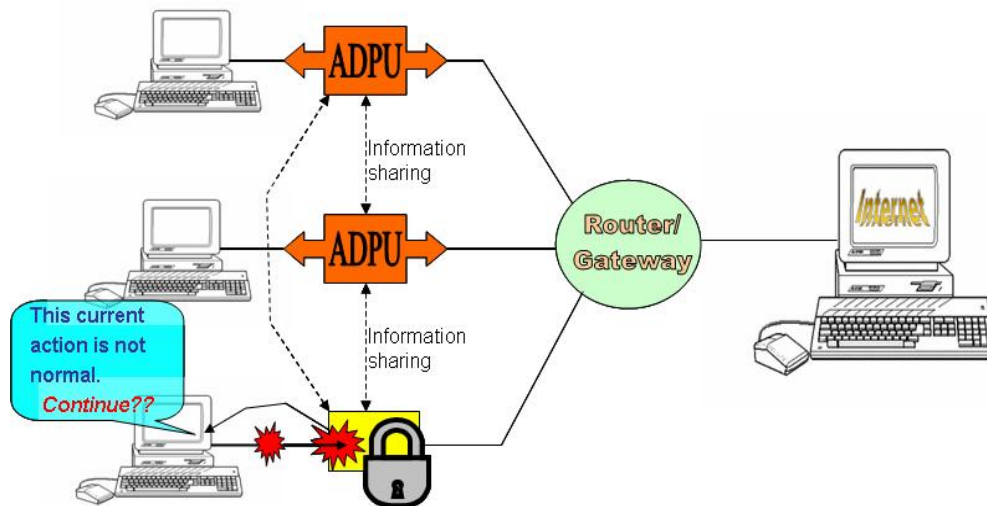


**Figure 22: Scenario 1 - Incoming Anomalous Activity**

Traffic is monitored from the end-user, but not allowed to connect to any node. If the end-user is infected with a worm, most worms try to spread themselves by connecting rapidly to several different addresses. This will be abnormal traffic and the ADPU can determine if the system is compromised. If the outbound connections are not new or abnormal, the ADPU assumes the system is not infected. It still continues to block traffic, however, until information is gathered from other ADPUs. The state of the node is set to “Susceptible,” whereas the ADPU is in an “Unknown” state. The ADPU is set to “Known” once other ADPUs confirm they have updated their monitoring policy for traffic from the source. Incoming traffic is monitored and either sent back or dropped. The end-user is disconnected from the rest of the network to prevent further infection if they are infected. Notifying the other ADPUs that have made recent connections with this “infected” system allows those ADPUs to set a stricter monitoring policy for

outbound traffic. Since the source address of the malicious activity is also sent to other ADPUs, traffic from this address is temporarily blocked.

Another scenario is abnormal activity traveling outbound from the end user as shown in Figure 23. The ADPU has not yet determined if the end-user is transmitting malicious activity or if this is a new desired connection. The current state of the ADPU is “Unknown” and the host is temporarily labeled as “Infected.” Traffic from the end-user is delayed or blocked until the ADPU can identify if the abnormal connection is safe. The user may be notified and given the opportunity to allow this new traffic to continue.



**Figure 23: Scenario 2 - Outbound Anomalous Activity**

If there is an infection such as a worm, it will attempt to connect outside the network, without acknowledging the returned validation message. A policy should be enforced similar to a password policy. If a given number of connection attempts are made without correct credentials, the status of the ADPU will be set to “Known” and the node will be labeled “Removed.” In the “Removed” state, traffic is blocked but not

monitored from the end-user. Pertinent information, such as IP address, protocol, and time stamp, about the connection is sent to neighboring ADPU's. Similar to the first scenario, the ADPU may not have detected the first attack packet. To prevent further spread of malicious packets, other systems with recent connections to this "infected" system should implement a stricter monitoring policy for outbound traffic. Inbound traffic is blocked to the "Removed" end-user until an administrator can determine if an attack was successful or the node has been cleared of malicious content.

Until all Anomaly Detection and Prevention Units have confirmed or updated their state to "Known," no ADPU's in the network can return to their default state of "Unknown." This does not mean the end-user's state returns to the default state (i.e., susceptible). If an end-user is still infected, the ADPU may return to "Unknown" but will still be blocking outbound traffic and set the end-user state to "Removed." This means the node is removed from the network until the node has been analyzed for malicious content and treated. An "Infected" system may return once an ADPU determines that no abnormal activity is occurring. The only way for a "Removed" node to return to normal operation is to send a special immune packet and credentials to the ADPU, or have an administrator physically reset the "fuse" on the ADPU.

A final scenario that will trigger the "fuse" to disconnect an end-user is a denial of service (DoS) attack. By sending a host abnormal activity or packets from multiple unknown sources, the ADPU would interpret these packets as a possible attack each time. This would disconnect the end-user from the network temporarily and if sent with the correct frequency, prevent the end-user from reconnecting to the network after the



ADPUs have determined there is no threat. If a DoS attack comes from a “trusted” source, the traffic will be passed to the end-user if not abnormal in size or format. If the trusted outside system has been infected and is performing a DoS on the end-user, it will most likely succeed by flooding the system with packets. To prevent this type of attack from occurring, the ADPU will be configured to determine the rate at which connections are being made to the end-user. As with worms, DoS attacks try to connect to a system as many times as possible to tie up the systems network resources [3]. To prevent DoS attacks, a system in the router/gateway itself should be employed to detect this type of activity and drop all packets related with it reliably.

## **5.5 System Components**

The ADPU device needs at least one RJ-45 10/100/1000 Mbps Ethernet port to connect between the detection device and the LAN switch. A second RJ-45 Ethernet port would be ideal to connect between the detection device and the end-user’s system, or a USB 2.0 connection would also work. In addition to these ports, it also requires either a PCMCIA interface or header pins to have a wireless connection card integrated. Current wireless protocols, such as 802.11n with MIMO (Multiple Input Multiple Output), will allow the highest range and data transfer capacity for the real-time ad-hoc network. This is required to establish a quality communication link between each device and update the network status at wired network speed.

For initial testing and setup purposes, a field programmable gate array (FPGA) should be used for easy hardware programmability, combined with the performance qualities of an application specific integrated circuit (ASIC). Real-time monitoring is an

essential part of packet inspection because maintaining data flow is essential. Post processing traffic is useful for examining how content is spread and what damage it will cause, but it is not an option for preventing malicious activity.

Several programmable boards are capable of performing the task of the ADPU while also having the ability to be upgraded for future additions. The datasheets for these boards can be found on their manufacturers websites and give detailed specifications of each. One such evaluation board that can handle the requirements of this project is Embedded Planet's EP82xxM version 1.0. The board's processor is a Freescale™ PowerQUICC™ II 82xx Processor and can operate in a stand-alone mode or plugged into a PCI bus [8]. This 32-bit reduced instruction set computer (RISC) processor includes an integrated PowerPC core. It is particularly well suited for target systems using PCI interfaces in networking infrastructure, telecommunications, and other embedded market applications. Features of the board, which make it suitable for the ADPU are: two fast Ethernet ports with their own physical addresses, an RS-232 monitoring port, RS-232 serial port, upgradeable SDRAM (up to 256 MB) , and upgradeable Flash memory (up to 64 MB). A second board that is similar to the EP82xxM is the EP8248E. It was designed to be inserted into a PCI slot, which could be beneficial if the ADPU were built into the network interface card or were used to replace the NIC [7].

Another board is the APS-ArmXF, powered by a Xilinx Virtex 1000E FPGA [36] with 1.5 million gates. The ARM block contains a 200 MHz 32-bit ARM processor with 32MB of SDRAM and 16MB of flash memory, upgradable to 1GB using the compact flash interface. This board has the option of stacking up to three modular XF-blocks,

yielding a system with a total of 4.5 million gates for future additions to the ADPU [1]. Although the ARM block only has one 10/100 Base-T Ethernet port, it does include two USB 2.0 ports with built in wireless LAN support to perform the ADPU's necessary functions. In addition to the hardware requirements, a small physical footprint is needed to be able to be placed inline with the end-users system and not take up workspace. This board's physical size is 3.8 inches by 4.5 inches, giving it the footprint size smaller than a CD case.

A FPIPS (Field Programmable Intrusion Protection System), could use an embedded version of Linux in a FPGA-based device with two Ethernet ports acting as a bridge between the node and its network. This bridge can act as a Firewall or even a reverse firewall, blocking traffic leaving a node and thus preventing an infection of other nodes. The device could use signature detection for known vulnerabilities (worms, Trojans, and viruses) while obtaining updated information from attacks performed on the Honeynet. With the FPGA acting as an Intrusion Prevention System (IPS), the status of the nodes on the network can trigger the actions in the FPGA by blocking outgoing ports or filtering incoming data more vigorously.

If the system were to be implemented, an embedded real-time operating system such as uClinux [5] or MicroC/OS-II [17] could run on a FPGA. Using this thin client system, an IDS (Intrusion Detection System) or a Honeypot is embedded in the FPGA between the rest of the network and the node, and acts as an early warning system for vulnerabilities and intrusions. The Honeypot/IDS alerts the system directly as well as surrounding systems. Information collected about vulnerability or how the hacker/worm

gained entry can be passed on to the other nodes of the network. With this information, systems could close these ports to deny access should the attack spread. Other options include, downloading a patch or checking online for security patches and fixes for the vulnerability. A detection and prevention system will not stop attacks, but will prevent the spread of an attack, once inside the local network. Antivirus and firewalls should still be used on each system to enhance security and if necessary, they may communicate with the device when they detect any activity, but a system similar to the one described above will allow early warning detection and prevention techniques to be used to safeguard systems and quarantine malicious attacks.

The honeynet would provide the added benefit of monitoring any unused IP addresses. Since the honeynet assumes the identity of a virtual computer at the unused IP address, this allows better detection of malicious attacks. If a pre-detection system is compromised, the attacker must still penetrate an actual system. By this time, the detection system will have alerted the actual system and its neighboring nodes of the attack, providing ample time to raise security policies, close unsecured ports, or turn on additional monitoring software to gain more knowledge about the attack.

## **5.6 Summary**

This chapter discusses the system behavior, policies, and components as well as what is needed to have successful and real-time wireless communication between devices. The basic behavior of this anomaly detection scheme originated from past research performed on the dynamics of virus and antivirus effects on a network. Expanding the network model from that research resulted in the multiple states the

ADPUs can assign the end-user systems and themselves to prevent an entire network from becoming infected. To fully monitor malicious activity, incoming as well as outgoing activity need to be watched. Abnormal activity leaving a machine is important to monitor because if malicious content has been introduced to the system through digital media instead of through the network, a user may not be aware their system is trying to spread the content to other systems on the network. Having the ADPU monitor both incoming and outgoing content enables a system to be quarantined from the network to prevent spreading or infection. It also allows for easy detection of which systems are infected with a malicious program or being used in a way that is not normal for the user, such as an internal threat.

## **VI. Conclusions**

### **6.1 Chapter Overview**

This chapter presents the conclusion of the research based on an extensive literature search, worm simulations, a network scenario simulation, and theoretical device implementation. This chapter also discusses the immediate and long-term impact of the techniques from computer vision, applied to a new network security device, would contribute to the protection of computer networks. In addition, recommendations are made to bring the theory of this proposed method to reality.

### **6.2 Conclusions of Research**

From studying other research systems and what is currently on the market (i.e. anti-virus, firewalls, and IDS systems), these system can usually indicate insidious agents before a widespread attack can gain momentum. The problem is, these systems fail to take advantage of widespread events considered “false alarms.” Simulating several networks still show that a worm can be a large threat to network traffic even with basic security measures in place. Encrypted traffic will not be detected with current systems that perform signature matching. Software solutions use system resources and if the operating system is already infected, the user may not know if the anti-virus or other threat protective software has been compromised. This would lead to false readings or even no detection of malicious activity at all. The implementation of a hardware solution would provide an additional layer of protection outside of the operating system environment and external to the host system.

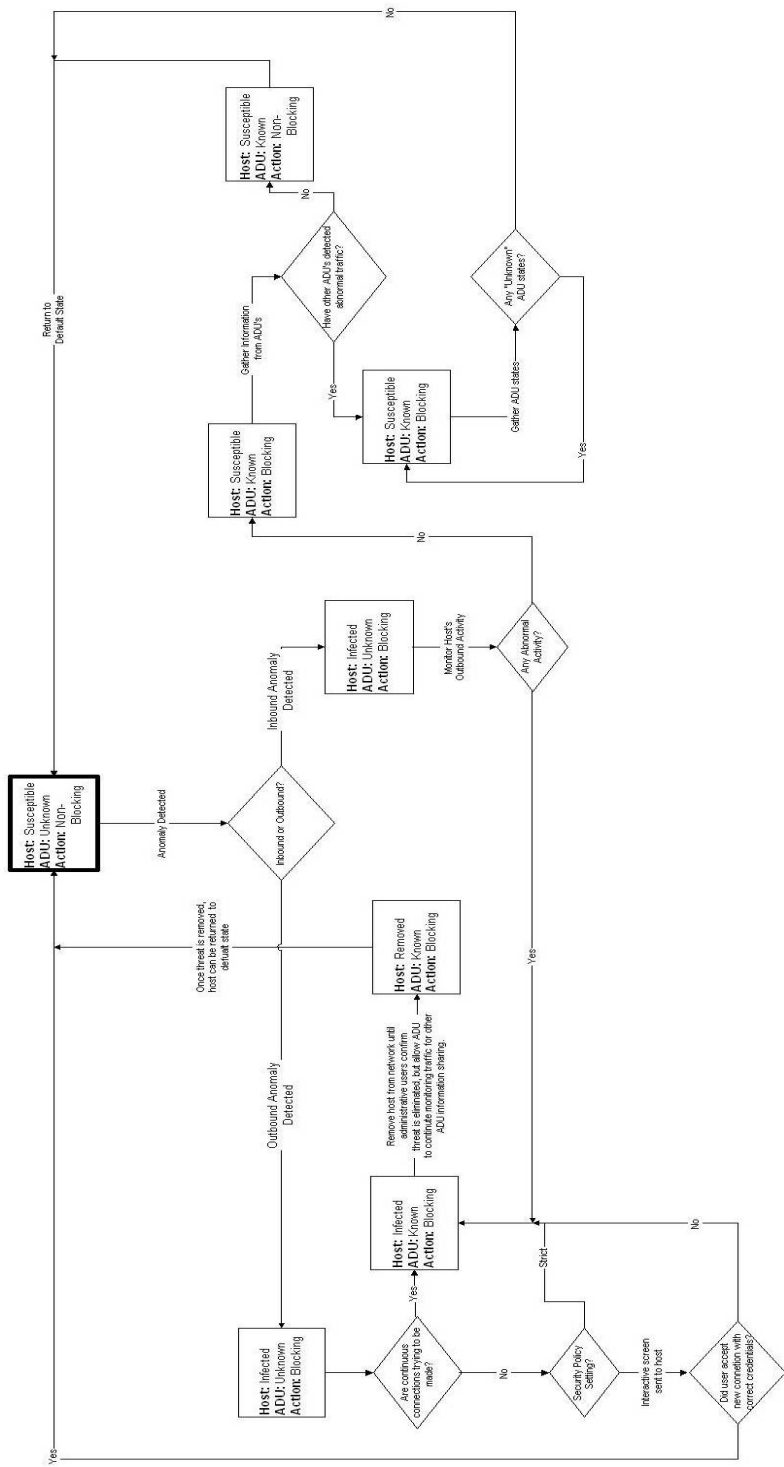
### **6.3 Significance of Research**

This research studied the application of iterative, decision fusion methods to be applied to predicting virus propagation. The work draws parallels between the two subjects and creates a bridge for cross-fertilization of ideas. In addition, the research studied the theoretical application an anomalous network traffic system. It identifies the components necessary for the network security system. By performing traffic analysis external to the end-users system, hardware will not have to be updated when newer operating systems are created. Reconfigurable hardware that monitors new connections and abnormal usage allows future network traffic to be encrypted for security reasons without designing a new hardware solution.

### **6.4 Summary**

Through the proven methods of computer vision and image analysis, this research used those methods as a basis and expanded upon them for network security. Predicting where an infection will spread to, is the next step in preventing a virus from encompassing a network. A theoretical system incorporating these methods was also studied and alleged to improved anomaly detection and threat preventative system for computer networks. Using the techniques described in this paper, along with the recommendations for hardware and setup, an inline distributive network monitoring device can be created to reduce future threats and the spreading of automated attacks within a network. A system as described by this research will be useful not only for threats entering a network, but also internal or insider threats.

Appendix 1: Full Flow Chart of ADPU System





## Bibliography

1. APS-ArmXF Rapid Development Platform. 2005,  
<http://www.associatedpro.com/armxf.html>.
2. Ballard, Dana H., and Christopher M. Brown. *Computer Vision*. Englewood Cliffs: Prentice-Hall Inc., 1982, pp. 84-89.
3. Chen, Shigang, and Qingguo Song. "Perimeter-Based Defense against High Bandwidth DDOS Attacks," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 16, No. 6, June 2005.
4. Deng, Hongmei, Qing-An Zeng, Dharma P. Agrawal, and Ravi Kothari. "A Light-Weight Network Intrusion Detection Approach Using Random Projection Technique," Center for Distributed and Mobile Computing, Dept. of ECECS University of Cincinnati, 2004.
5. Embedded Linux Microcontroller Project –  $\mu$ Clinux. 2005,  
<http://www.uclinux.org/index.html>
6. Fischler, Martin and Oscar Firschein. *Readings in Computer Vision*. San Francisco: Morgan Kaufmann Publishers, Inc., 1987, pp. 585-605.
7. Freescale PowerPC PowerQUICC II 8248 EP8248E PCI Edge Single Board Computer, 2005, <http://www.embeddedplanet.com/products/ep8248.asp>.
8. Freescale PowerQUICC II PowerPC (MPC8280, MPC8270, MPC8266, MPC8265 or MPC8250) PrPMC Single Board.2005,  
<http://www.embeddedplanet.com/products/ep8280.asp>.
9. Garetto, Michele, Weibo Gong, and Don Towsley. "Modeling Malware Spreading Dynamics," IEEE Infocom 2003 (0-7803-7753-2/03).
10. Höglund, Albert J., Kimmo Hätönen, and Antti S. Sorvari. "A Computer Host-Based User Anomaly Detection System Using the Self-Organizing Map," *IJCNN*, Vol. 05, No. 5, p. 5411, IEEE-INNS-ENNS 2000. (0-7695-0619-4/00).
11. Horn, Berthold Klaus Paul. *Robot Vision*. Cambridge: The MIT Press, 1986, pp. 259-265.
12. Hu, Qiang, Xi Zhang, and Debanjan Saha. "Modeling Virus and Anti-Virus Dynamics in Topology-Aware Networks," *IEEE Communications Society, Globecom* 2004. (0-7803-8794-5/04).

13. Hutchings, B. L., and R. Franklin and D. Carver. "Assisting Network Intrusion Detection with Reconfigurable Hardware." Department of Electrical and Computer Engineering, Brigham Young University. FCCM'02 IEEE. 2002.
14. Kobrick, Jonathan. "NIDS Countermeasures: What, Why, Where, When, and How," SANS Institute, GSEC Option 1 version 1.4b, March 2003.
15. Kolesnikov, Oleg, and Wenke Lee. "Advanced Polymorphic Worms: Evading IDS by Blending in with Normal Traffic," College of Computing Georgia Institute of Technology, Atlanta, GA 30332. 2004.
16. Kruegel, Christopher, and Vigna Giovanni. "Anomaly Detection of Web-based Attacks," CCS'03, 2003. (ACM 1-58113-783-9/03/0010).
17. Lambrosse, Jean J. *MicroC/OS-II The Real-Time Kernel*. Lawrence: CMP Books, CMP Media, Inc., 1999.
18. Levine, John, Richard LaBella, Henry Owen, Didier Contis, and Brian Culver. "The Use of Honeynets to Detect Exploited Systems Across Large Enterprise Networks," Proceedings of the 2003 IEEE, Workshop on Information Assurance, United States Military Academy, West Point, NY, June 2003.
19. Lin, Jessica, and Dimitrios Gunopulos. "Dimensionality Reduction by Random Projection and Latent Semantic Indexing," Department of Computer Science & Engineering, University of California, Riverside, 2002.
20. Lockwood, John W., James Moscola, Matthew Kulig, David Reddick, and Tim Brooks. "Internet Worm and Virus Protection in Dynamically Reconfigurable Hardware," *Military and Aerospace Programmable Logic Device (MAPLD)*, Washington DC, 2003. Paper E10, Sep 9-11, 2003.
21. Matzner, Sara, and Tom Hetherington. "Detecting Early Indications of a Malicious Insider," IAnewsletter, Vol. 7, Ed. 2, 2004
22. Nazario, Jose. *Defense and Detection Strategies against Internet Worms*. Norwood: Artech House Inc. 2004.
23. Nazario, Jose, Jeremy Anderson, Rick Walsh, and Chris Connelly. "The Future of Internet Worms". Crimelabs Research, July 2001, <http://www.crimelabs.net/docs/worms/worm.pdf>.
24. Newman, M.E.J., Stephanie Forrest, and Justin Balthrop. "Email networks and the spread of computer viruses," The American Physical Society, Ed.66, 2002.

25. Pelillo, Marcello and Mario Refice. "Learning Compatibility Coefficients for Relaxation Labeling Process," *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 16, No. 9, September 1994.
26. Prager, J.M. "Extracting and labeling boundary segments in natural scenes." *IEEE Trans. PAMI* 2, p. 16-27, January 1980.
27. Rosenfeld, Azriel, and Avinash C. KAK. *Digital Picture Processing Second Edition Volume 2*. Orland: Academic Press, Inc., 1982.
28. Sink, Michael, P.E. "The Use of Honeypots and Packet Sniffers for Intrusion Detection," *GIAC Security Essentials*, version 1.2b, April 15, 2001.
29. Spitzner, Lance. "Honeypots - Definitions and Value of Honeypots," 2003, <http://www.tracking-hackers.com/papers/honeypots.html>. Last accessed 18 Jan. 2005.
30. Symantec Research Labs. "Worm Simulator Program", Cupertino, CA, 2005
31. Thottan, Marina, and Chuanyi Ji. "Anomaly Detection in IP Networks," *IEEE Transactions on Signal Processing*. Vol. 51, No. 8, August 2003.
32. U.S. vs. Morris: Court document 928 F.2d 504, 59 U.S.L.W 2603.
33. Wang, Haining, Danlu Zhang, and Kang G. Shin. "Change-Point Monitoring for the Detection of DoS Attacks," *IEEE Transactions on Dependable and Secure Computing*, Vol.1, No.4, 2004.
34. Williamson, Matthew M. "Throttling Viruses: Restricting propagation to defeat malicious mobile code," Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC.02), 2002.
35. Willmann, Paul, Hyong-youb Kim, Scott Rixner, and Vijay S. Pai. "An Efficient Programmable 10 Gigabit Ethernet Network Interface Card," International Symposium on High-Performance Computer Architecture (HPCA), San Francisco, CA, February 2005.
36. XESS homepage announcements. 2005, <http://www.xess.com/index.html>.
37. Yurcik, William, David Loomis, Alexander D. Korzyk, Sr. "Predicting Internet Attacks: On Developing An Effective Measurement Methodology," Proceedings of the 18th Annual International Communications Forecasting Conference (ICFC-2000) Seattle WA. USA, September 2000.

38. Winston, Patrick H. *Artificial Intelligence 3rd ed.* Reading: Addison-Wesley Publishing Co., 1992, pp. 249-266.
39. Zalewski, M. "Writing Internet Worms for Fun and Profit," 2000.  
<http://lcamtuf.coredump.cx/worm.txt>.

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 074-0188	
<p>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.</p> <p><b>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</b></p>					
1. REPORT DATE (DD-MM-YYYY)		2. REPORT TYPE		3. DATES COVERED (From – To)	
13-06-2006		Master's Thesis		June 2004 – April 2006	
4. TITLE AND SUBTITLE  An Iterative Relaxation Approach for Anomaly Detection and Preventive Measures in Computer Networks				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S)  Bell, Garrick A., Civilian, DAGSI				5d. PROJECT NUMBER If funded, enter ENR #	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way, Building 640 WPAFB OH 45433-8865				8. PERFORMING ORGANIZATION REPORT NUMBER  AFIT/GCE/ENG/06-01	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT  APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT  It is proposed to develop a framework of detecting and analyzing small and widespread changes in specific dynamic characteristics of several nodes. The characteristics are locally measured at each node in a large network of computers and analyzed using a computational paradigm known as the Relaxation technique. The goal is to be able to detect the onset of a worm or virus as it originates, spreads-out, attacks and disables the entire network. Currently, selective disabling of one or more features across an entire subnet, e.g. firewalls, provides limited security and keeps us from designing high performance net-centric systems. The most desirable response is to surgically disable one or more nodes, or to isolate one or more subnets.  The proposed research seeks to model virus/worm propagation as a spatio-temporal process. Such models have been successfully applied in heat-flow and evidence or gestalt driven perception of images among others. In particular, we develop an iterative technique driven by the self-assessed dynamic status of each node in a network. The status of each node will be updated incrementally in concurrence with its connected neighbors to enable timely identification of compromised nodes and subnets. Several key insights used in image analysis of line-diagrams, through an iterative and relaxation-driven node labeling method, are explored to help develop this new framework.					
15. SUBJECT TERMS Computer Viruses, Computer Networks, Relaxation, Network Security, Data Processing Security					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Guna Seetharaman, ENG/GCE
U	U	U	UU	117	19b. TELEPHONE NUMBER (Include area code) (937) 255-3636, ext 4612 (Guna.Seetharaman@afit.edu)

