An Immunological Model of Distributed Detection and Its Application to Computer Security

By

Steven Andrew Hofmeyr

B.Sc. (Hons), Computer Science, University of the Witwatersrand, 1991
M.Sc., Computer Science, University of the Witwatersrand, 1994

Doctor of Philosophy
Computer Science

May 1999

Dedication

To the Babs for having such patience when I was so far away, and to my dearest Folks for getting me this far.

Acknowledgments

The author gratefully acknowledges the help of the following people: D. Ackley, P. D'haeseleer, S. Forrest, G. Hunsicker, S. Janes, T. Kaplan, J. Kephart, B. Maccabe, M. Oprea, B. Patel, A. Perelson, D. Smith, A. Somayaji, G. Spafford, and all the people in the Adaptive Computation Group at the University of New Mexico.

This research was supported by the Defense Advanced Research Projects Agency (grant N00014-96-1-0680) the National Science Foundation (grant IRI-9711199), the Office of Naval Research (grant N00014-99-1-0417), the IBM Partnership award, and the Intel Corporation.

STEVEN HOFMEYR

The University of New Mexico
May 1999

An Immunological Model of Distributed Detection and Its Application to Computer Security

By

Steven Andrew Hofmeyr

Doctor of Philosophy
Computer Science

May 1999

An Immunological Model of Distributed Detection and Its Application to Computer Security

by

Steven Andrew Hofmeyr

B.Sc. (Hons), Computer Science, University of the Witwatersrand, 1991
M.Sc., Computer Science, University of the Witwatersrand, 1994
Ph.D., Computer Science, University of New Mexico, 1999

Abstract

This dissertation explores an immunological model of distributed detection, called *negative detection*, and studies its performance in the domain of intrusion detection on computer networks. The goal of the detection system is to distinguish between illegitimate behaviour (*nonself*), and legitimate behaviour (*self*). The detection system consists of sets of *negative detectors* that detect instances of nonself; these detectors are distributed across multiple locations. The negative detection model was developed previously; this research extends that previous work in several ways.

Firstly, analyses are derived for the negative detection model. In particular, a framework for explicitly incorporating distribution is developed, and is used to demonstrate that negative detection is both scalable and robust. Furthermore, it is shown that any scalable distributed detection system that requires communication (memory sharing) is always less robust than a system that does not require communication (such as negative detection). In addition to exploring the framework, algorithms are developed for determining whether a nonself instance is an undetectable *hole*, and for predicting performance when the system is trained on non-random data sets. Finally, theory is derived for predicting false positives in the case when the training set does not include all of self.

Secondly, several extensions to the model of distributed detection are described and analysed. These extensions include: multiple representations to overcome holes; activation thresholds and sensitivity levels to reduce false positive rates; costimulation by a human operator to eliminate autoreactive detectors; distributed detector generation to adapt to changing self sets; dynamic detectors to avoid consistent gaps in detection coverage; and memory, to implement signature-based detection.

Thirdly, the model is applied to network intrusion detection. The system monitors TCP traffic in a broadcast local area network. The results of empirical testing of the model demonstrate that the system detects real intrusions, with false positive rates of less than one per day, using at most five kilobytes per computer. The system is tunable, so detection rates can be traded off against false positives and resource usage. The system detects new intrusive behaviours (anomaly detection), and exploits knowledge of past intrusions to improve subsequent detection (signature-based detection).

Contents

List of Figures

List of Tables

Glossary of Symbols

The symbols are listed in the order in which they appear in the text.

ℓ String length

r Match threshold

U Universe

S Self set

N Nonself set

Γ Kolmogorov complexity

$\mathcal{D}$ Detection system

f Binary classification function

$\mathcal{M}$ Detection system memory

U_{test} Test set

U_{trn} Training set

ε^{+} False positive error

ε^{-} False negative error

L Set of locations

n Number of locations

M_l Memory capacity at location l

g Global classification function

ε_g^{+} Global false positive

ε_g^{-} Global false negative

k Constant

Λ Representation function

U_R Representation of U

h Match rule

C_d Cover set of detector d

p_{contig} Probability of matching under the contiguous bits rule

p_{hamm} Probability of matching under the Hamming match rule

F_i The event that d does not match any s in S_R

p_M Probability of a match

$V(d)$ Valid detector d

ρ Number of retries in detector generation

η_l Number of detectors at location l

p_{ε^-} Probability of a false negative error

p_{ε^+} Probability of a false positive error

θ Algorithm which computes if a given nonself string is a hole

ϕ The overlap function

Y Decimal value of binary string remapped by the linear congruential operator

α Parameter for the linear congruential operator

β Parameter for the linear congruential operator

X Discrete random process

X_i Random variable of X at time-step i

$\tilde{p}(s)$ Sample distribution of self set

$p(s)$ Distribution of self set

$m(t)$ Number of unique self strings that occur in t time-steps

n_L Number of locations

n_d Number of detectors

Φ Parameter set for a representation

S_{trn} Training set

S_{test} Test set, self strings only

N_{test} Test set, nonself strings only

c_d Match count for detector d

τ Activation threshold

γ_{match} Match decay probability

σ_l Sensitivity level at location l

ω Effect of sensitivity

γ_ω Sensitivity decay probability

$\Delta\varepsilon^+$ False positive error rate

$\Delta\varepsilon^-$ False negative error rate

ψ Detection rate

β Constant used when predicting retries versus detection rate

$H(a, b)$ Hamming distance between binary strings a and b

ψ_{all} Detection rate over all strings in a nonself incident

$\psi_{nonself}$ Detection rate over only the nonself strings in a nonself incident

a Offset in power law distribution of self set

b Exponent in power law distribution of self set

R Ratio of increase in self strings when size of self increases

T_s Costimulation delay

T Tolerization period

λ_A Queue arrival rate

λ_D Queue departure rate

R Ratio of queue arrivals to departures

p_i Probability of occurrence of a self string, s_i

Y Random variable for the number of matches in a queue

t Match decay period

p_{death} Probability of detector death

ι Fraction of immature detectors

$E(\text{life})$ Expected lifetime

m_d Maximum number of memory detectors

Chapter 1

Introduction

The Immune System (IS) is complex, and to the observer, has novel solutions for solving real-world problems. We can apply this wealth of evolved solutions to systems design if we can find an artificial system that faces similar problems to those faced by the IS. To do this we need to have a reasonable understanding of immunology.

1.1 Immunology

From a teleological viewpoint, the IS has evolved to solve a particular problem. Fundamentally, such a viewpoint is wrong, because the IS is not necessarily a minimal system (there may be simpler ways to solve the same problem) but this viewpoint is useful for expository purposes: it is easier to understand the IS to a first approximation if the components and mechanisms are viewed with the assumption that they exist to solve a particular problem.

The human body is under constant siege by a plethora of inimical micro-organisms such as bacteria, parasites, viruses, and fungi, known collectively as pathogens. These pathogens are the source of many diseases and ailments, for example, pneumonia is caused by bacteria, AIDS and influenza are caused by viruses, and malaria is caused by parasites. Pathogens in particular can be harmful because they replicate, leading to a rapid demise of the host if left unchecked. In addition to micro-organisms, the human body is threatened by toxic substances that can do serious harm if they are not cleared from the body. In this dissertation it is assumed that the "purpose" of the IS is to protect the body from the threats posed by pathogens, and to do so in a way that minimizes harm to the body and ensures its continued functioning[1].

There are two aspects to the problem that the IS faces: the identification or *detection* of pathogens, and the efficient *elimination* of those pathogens while minimizing harm to the body, from both pathogens and the IS itself. The detection problem is often described as that of distinguishing "self" from "nonself" (which are elements of the body, and pathogens/toxins, respectively). However, many pathogens are not harmful, and an immune response to eliminate them may damage the body. In these cases it would be healthier not to respond, so it would be more accurate to say that the problem faced by the IS is that of distinguishing between *harmful* nonself and everything else [Matzinger, 1994, Matzinger, 1998][2]. Once pathogens have been detected, the IS must eliminate them in some manner. Different pathogens have to be eliminated in different ways, and the components of the IS that accomplish this are called effectors. The elimination problem facing

[1] This is a limited view of "purpose"; in general, it may be that the purpose of the immune system is to maintain homeostasis, which includes protecting the body from pathogens that could disrupt that homeostasis.

[2] However, the fact that the IS does react to "harmless" pathogens is essential to immunization (see section 2.1.3).

the IS is that of choosing the right effectors for the particular kind of pathogen to be eliminated.

1.2 Computer Security

Phrased this way, the problem that the IS addresses is similar to the problem faced by computer security systems: the immune system protects the body from pathogens, and analogously, a computer security system should protect computers from intrusions. This analogy can be made more concrete by understanding the problems faced by computer security systems. There are several aspects to computer security [Meade, 1985, Garfinkel & Spafford, 1996]:

Confidentiality: Access to restricted or confidential data should only be allowed to authorized users, for example, it is imperative for military institutions to limit knowledge of classified information.

Integrity: Data should be protected from corruption, whether malicious or accidental. In some cases, it is essential to preserve the integrity of critical information, for example, there should be no tampering with information used by emergency services.

Availability: Both information and computer resources should be available when needed by legitimate users. In particular, this is essential in cases where such information is needed to make critical decisions within a limited time, for example, in air-traffic control.

Accountability: In the case where the compromise of a computer system has been detected, the computer security system should preserve sufficient information to be able to track down and identify the perpetrators.

Correctness: False alarms from incorrect classification of events should be minimised for the system to be usable. Low levels of correctness can interfere with other aspects of security, for example, availability will be reduced if a user's legitimate actions are frequently labeled as alarms, and so not permitted.

The importance of these aspects of computer security depends on the security policy for the computer system. The policy is a description or definition of what activities are and are not allowed, by the different users and software components of the system. Policy must first be specified by those in charge of the system, and then implemented in some form. Both specification and implementation are prone to error, being subject to the same limitations as program verification and implementation: programs are not verifiable in general, and implementation is always subject to error.

It is generally agreed that implementing and maintaining secure computer systems is difficult, in that we have no way of ensuring that a certain level of security has been achieved [Frank, 1994, Crosbie & Spafford, 1994, Kumar & Spafford, 1994, Lunt, 1993, Anderson, *et al.*, 1995, Blakely, 1997]. Security holes are exploited by intruders breaking into systems, or by viruses or worms. Such holes are often the result of faults or design flaws in system or application software, or in the specification or implementation of security policies. Even if it were possible to design and build a completely secure system, the investment in systems deployed in the 1990s makes it infeasible to replace every existing system. Furthermore, the continual updating of old systems, and the addition of new components will continue to produce novel vulnerabilities.

The similarity between the problem of computer security and that faced by the IS can be shown by translating the language of immunology into computer security terms: we can say that the IS detects abuses of an implicitly specified policy, and responds to those abuses by counter-attacking the source of the abuse. The policy is implicitly specified by natural selection, and emphasises only some aspects of security:

availability and correctness are of paramount importance, and to a lesser extent, integrity and accountability. Availability means enabling the body to continue functioning under an onslaught of pathogens; correctness means preventing the IS from attacking the body, (i.e., minimising auto-immune disorders); integrity means ensuring that the genes that encode for cell functions are not corrupted by pathogens; and accountability means finding and destroying the pathogens responsible for illness[3]. The one aspect of security that is not important to the IS is confidentiality: there is no notion of secret or restricted data in the body that must be protected at all costs from outsiders (e.g., we continuously shed cells with our DNA in them.

The IS is analogous to a computer security system, one that is designed to safeguard against breaches in an implicit policy. However, the architecture of the IS is different from that of the computer security systems of the 1990s. The first layer of defense in these computer security systems is provided by static access mechanisms, such as passwords and file permissions. Although essential, these access mechanisms are either too limited to provide comprehensive security, or are overly restrictive for legitimate users of the computer system. Several layers have been added on to the original defenses, some of the most important of these being cryptography [Denning, 1992], which is used for implementing secure channels and host authentication, and firewalls [Chapman & Zwicky, 1995], which provide another layer of defense in a networked system by filtering out undesirable network traffic. Yet another layer of defense is provided by dynamic protection systems that detect and prevent intrusions. These dynamic protection systems are known as Intrusion Detection (ID) systems [Anderson, 1980, Denning, 1987].

These computer security systems fall short of what could be accomplished: in a survey carried out by the Computer Security Institute in collaboration with the Federal Bureau of Investigation, 64% of 520 computer security practitioners surveyed reported security breaches during the 1998 financial year, a 16% increase from the year before [Power, 1998]. Only half of the respondents could estimate their financial losses from these incidents, at about 138 million dollars. According to [Power, 1998], we should assume that these estimates (both of losses and intrusions) are conservative, because many institutions will not be aware they have been compromised, and of those who become aware, few will report it. However, with all these dire figures, it is worth noting that only 35% of the respondents used ID systems. It is not clear why this is the case, whether it is that the current ID systems are not cost-effective, or simply that they are an innovation that has not yet been widely accepted.

There is some indication that current ID systems are not effective enough, and suffer from lack of correctness. In an evaluation performed by Lincoln Laboratory in 1998, ID systems detected 50 to 70% of attacks with false alarm rates of between one and ten per day [Lippman, 1998]. Although these false alarm rates are acceptable according to [Lippman, 1998], the detection rates can be improved, and in particular, none of the systems tested was able to detect novel new intrusions. Most of these systems carried out signature-based[4] detection, meaning that they stored patterns of known intrusive behaviour and then scanned for occurrences of those patterns. Few systems carried out anomaly detection, where the incidents are unknown ahead of time (that is, they are not included in a training set), and detection is a process of scanning for deviations from a known normal behaviour. These systems could not detect novel intrusions because they did not perform anomaly detection.

By contrast, the IS makes use of both signature-based and anomaly detection. It has mechanisms for detecting deviations from a set of normal patterns, and it has ways of storing and recalling specific patterns associated with previous pathogenic attacks. It functions effectively[5] in a distributed environment where there is competition for limited resources, against adversaries that are evolving to evade the IS. Thus the IS can indicate ways in which we can improve our existing ID systems. This, then, is the topic of this dissertation: demonstrating how immunology can help us design better ID systems, ones that carry out both signature-

[3] Accountability in the IS is limited to within the confines of the body.

[4] Also misleadingly called misuse detection. The term "signature-based" is used throughout this dissertation.

[5] Without immune systems, we would not survive for more than a few weeks.

based and anomaly detection.

1.3 Principles for an Artificial Immune System

The crossover between biology and computer science can be fruitful for both disciplines: computers can be used to model biological systems to improve our understanding of those systems, and we can use an understanding of the mechanisms underlying biological systems to improve the way we design computer systems. Here, the focus is on the latter case: using biological metaphors to enable us to build better computer systems.

There are several levels at which we can use biological metaphors [Somayaji, 1998]: buzzwords, architectures, algorithms and principles. Buzzwords can be a useful guide to thinking, but in general they do not suggest concrete ways of improving systems. We need to extract more information from biological systems. This dissertation focuses on biological principles, architectures, and algorithms extracted from immunology and applied to the design of systems, specifically a system for detecting intrusions in networks of computers.

The design of the ID system presented in this dissertation is guided by the following principles [Somayaji, *et al.*, 1997]:

Distributed protection: The IS consists of millions of agents or components, distributed throughout the body. These components interact locally to provide protection in a completely distributed fashion: there is little or no centralized control or coordination, and hence no single point of failure in the system.

Diversity: There are several forms of diversity in the immune system. For example, each individual in a population has a unique IS, which improves the robustness of a population, because all individuals will not be vulnerable to the same extent to the same diseases. Within an individual, the components of the IS are different, providing diverse pattern recognition so that the IS can detect a variety of pathogens.

Robustness: Individual components of the IS are multitudinous and redundant, and the architecture of the IS is such that the loss of only a few of these components will have little effect on the functioning of the IS. These *disposable* components combined with the lack of centralized or hierarchical control in the IS make it robust and tolerant of faults in the form of failures in a few components.

Adaptability: The IS can adapt to pathogenic structures, "learning" to recognize pathogens with increasing accuracy so that it can mount an increasingly effective response. Adaptation speeds up a current response and enables future responses to be more rapid than past responses (see the next point, memory).

Memory (signature-based detection): Adaptations remain in force for periods of time up to the lifetime of an organism. This immunological "memory" allows the IS to more rapidly the second time around to pathogens similar to ones it has encountered in the past. Immune memory implements signature-based detection, by allowing the IS to monitor for characteristics of known pathogens.

Implicit policy specification: The definition of self used by the IS is empirically defined. Self is implicitly determined by the "monitoring" of proteins that are currently in the body. The advantage of this approach is that self is equivalent to the actual normal behaviour, and not a specification of what normal behaviour *ought to be*. In other words, the IS does not suffer from the problems inherent in trying to correctly specify a security policy.

Flexibility: The IS is flexible in the allocation of resources for the protection of the body. When a serious infection threatens the body, the IS draws upon more resources, generating more IS components, and at other times, the IS uses fewer resources. For example, a stressed person is more vulnerable to illness, and an ill person will generally be better off resting (and thus freeing more resources for the IS). Furthermore, the IS is flexible in where it directs its resources: the more a particular pathogen type is responsible for damage, the more likely it is that the IS will tailor its response to that pathogen type.

Scalability: Viewed from the perspective of distributed processing, the IS is scalable: communication and interaction between all IS components is localized, so there is little overhead when increasing the number of components. There may be architectural constraints on this scalability, but at the level of individual components, it is expected that resource expenditure would scale linearly with the number of components.

Anomaly detection: Through a variety of mechanisms, the immune system has the ability to detect novel pathogens (pathogens to which it has not been previously exposed). In most cases this anomaly detection is not as effective as signature-based detection (it is the efficacy of a primary response as compared to a secondary response, see section 2.1.3), but anomaly detection is essential to the survival of the organism, because any organism will always encounter novel pathogens in its lifetime.

These principles can be regarded as general guidelines for design which can, in general, be achieved by using algorithms or mechanisms copied directly from immunology. However, there will be times when the immunological algorithms or mechanisms are not suitable, and in those cases new algorithms will be required. It is worth emphasizing that for computer security applications the primary concern should not be exact mimicry of the IS in all its details; rather, the analogy should only be followed to the extent that it proves useful.

1.4 The Contributions of this Dissertation

There are three primary contributions made by this dissertation.

1. Formalization and new analyses of the immunological model of distributed detection, originally developed in [Forrest, *et al.*, 1994].
2. Description and empirical testing of several extensions to the basic model of distributed detection.
3. Application of the model of distributed detection to network ID.

These contributions are restated in more detail in section 7.2.

1.5 The Remainder of this Dissertation

The next chapter (2) describes background material necessary for an understanding of the work that follows, including an overview of salient features of immunology, a description of network security and network ID systems, and a brief look at the immunological model of distributed detection that is the basis of the results in this dissertation. Chapter 3 presents the immunological model of distributed detection formally and in more detail, adds additional features, and explores properties of the model. The model is then applied to the domain of network ID, and the results of implementation and experimentation are presented in chapter 4. Further extensions to the model in the context of network ID are described in chapter 5, and the results of experiments testing these extensions are presented. The penultimate chapter (6) presents various implications of the research, including ways of incorporating automated response, different domains that the model could be applied to, and perspectives for computer science and immunology. The dissertation is concluded by chapter 7.

Chapter 2

Background

This dissertation investigates a model of distributed detection based on the principles, algorithms and architecture of the immune system, and applies this model to the design of a new computer security system, specifically, an intrusion detection system that monitors traffic on a network of computers. The key word here is *detection*: the issue of intrusion response is not investigated, except to give some ideas for an automated response system in chapter 6. This chapter reviews all material necessary for understanding the remainder of the dissertation. This includes a brief overview of immunology, a description of a first attempt that used immunology for the design of an ID system, an overview of network ID in general, and details of previous work in defining a framework for distributed anomaly detection using immunological algorithms. Because of the large volume of background material, only information relevant to the dissertation is described; hence the overview of immunology is limited[1], and the survey of ID systems is focused on those applied to network traffic.

2.1 Immunology for Computer Scientists

The ability of the immune system to detect and eliminate most pathogens is essential for survival; without an IS we die within weeks. The architecture of the IS is multi-layered, with defenses on several levels. Most elementary is the skin, which is the first barrier to infection. Another barrier is physiological, where conditions such as pH and temperature provide inappropriate living conditions for foreign organisms. Once pathogens have entered the body, they are dealt with by the innate immune system and by the acquired or adaptive immune system. The innate immune system primarily consists of the endocytic and phagocytic systems, which involve motile scavenger cells such as macrophages that ingest extracellular molecules and materials, clearing the system of both debris and pathogens. Most of the inspiration for this research has been drawn from the adaptive IS, and so this overview is focused on adaptive immunity.

The adaptive immune system is so-called because it adapts or "learns" to recognize *specific* kinds of pathogens, and retains a "memory" of them for speeding up future responses. The learning occurs during a primary response to a kind of pathogen not encountered before by the IS. The primary response is slow, often first only becoming apparent ninety-six hours after the initial infection, and taking up to three weeks to clear the infection. After the primary response clears the infection, the IS retains a memory of the kind of pathogen that caused the infection. Should the body be infected again by the same kind of pathogen, the IS does not have to re-learn to recognize the pathogens, because it "remembers" their specific appearance, and so can

[1] In [Hofmeyr, 1998] a more detailed overview of immunology is given, but one that is still accessible to non-immunologists. For detailed references, the interested reader should consult [Piel, 1993, Janeway & Travers, 1996, Paul, 1989].

mount a much more rapid and efficient secondary response. The secondary response is usually quick enough so that there are no clinical indications of a re-infection. Immune memory can confer protection up to the life-time of the organism (a canonical example is measles). The adaptive immune system is also known as the acquired immune response system, because it is responsible for immunity which is adaptively *acquired* during the life-time of the organism.

The adaptive immune system primarily consists of certain types of white blood cells, called lymphocytes, which cooperate to detect pathogens and assist in the elimination of those pathogens. Although lymphocytes play a role in the elimination of pathogens, we can abstractly view them as mobile, independent *detectors*, which circulate around the body via the blood and lymph systems. There are millions of these lymphocytes, forming a system of distributed detection, where there is no centralized control, and little, if any, hierarchical control. Detection and elimination of pathogens is a consequence of millions of cells - detectors - interacting through simple, localized rules, to give rise to an efficient,[2] distributed system.

2.1.1 Recognition

A detection or recognition event occurs in the IS when chemical bonds are established between receptors on the surface of an immune cell, and epitopes, which are locations on the surface of a pathogen or protein fragment (a peptide). Both receptors and epitopes have complicated three-dimensional structures that are electrically charged. The more complementary the structure and charge of the receptor and the epitope, the more likely it is that binding will occur. See figure 2.1. The strength of the bond between a receptor and an epitope is termed the affinity. Consequently, receptors are *specific* because they bind only to a few similar epitope structures or patterns. This specificity extends to the lymphocytes themselves: receptor structures may differ between lymphocytes, but on a single lymphocyte, all receptors are identical, making a lymphocyte specific to a particular set of similar epitope structures (this feature is termed *monospecificity*). Pathogens often have multiple, different epitopes, reflecting their molecular structures, so several different lymphocytes may be specific to a single kind of pathogen.

A lymphocyte has on the order of 10^5 receptors on its surface, all of which can bind epitopes. Having multiple identical receptors has several beneficial effects. Firstly, it allows the lymphocyte to "estimate" the affinities of its receptors for a given kind of epitope, through frequency-based sampling: as the affinities increase, so the number of receptors binding will increase. The number of receptors that bind can be viewed as an estimate of the affinity between a single receptor and an epitope structure[3]. Secondly, having multiple receptors allows the lymphocyte to estimate the number of epitopes (and thus infer the number of pathogens) in its neighbourhood: the more receptors bound, the more pathogens in the neighbourhood. Finally, monospecificity is essential to the immune response, because if lymphocytes were not monospecific, reaction to one kind of pathogen would induce response to other, unrelated epitopes.

The behaviour of lymphocytes is dictated by affinities: when the number of receptors bound exceeds some threshold, the lymphocyte is activated (this can be termed a "detection event")[4]. Thus, a lymphocyte will only be activated by pathogens if its receptors have sufficiently high affinities for particular epitope structures on the pathogens, and if the pathogens exist in sufficient numbers in the locality of the lymphocyte.

[2]Obviously, the definition of "efficient" is relative; it cannot be claimed that the architecture of the IS is optimal in any sense because it is not clear that evolution optimizes anything, and furthermore, if it does, it is not clear what is being optimized and over what time scale. However, the IS possesses many qualities that our artificial systems lack, and it implements these in a way that is efficient when compared to our artificial systems.

[3]Of course, this is an idealization, because the receptors may bind different epitope structures, so what is being "estimated" is a rather arbitrary mean of the affinities between the receptors and different epitope structures. However, as the affinities increase, the estimation becomes more accurate because the epitope structures must be increasingly similar.

[4]This is an expository simplification; lymphocytes require additional signals to be activated. This is termed costimulation, and is discussed in section 2.1.4.

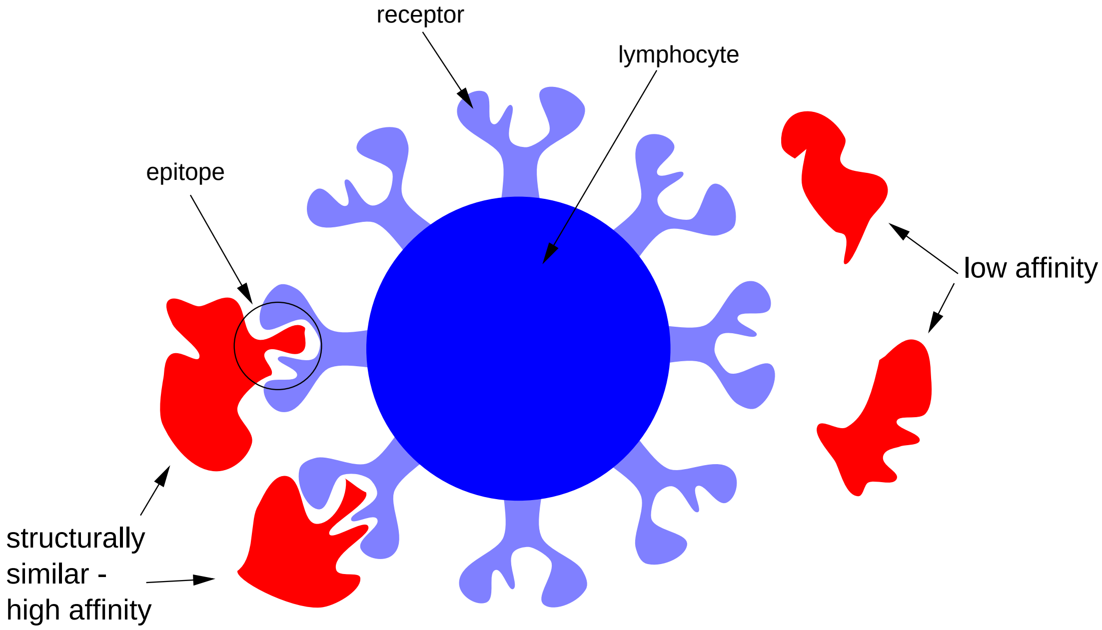

Figure 2.1: Detection is a consequence of binding between complementary chemical structures. The surface of a lymphocyte is covered with receptors. The pathogens on the left have epitope structures that are complementary to the receptor structures and so the receptors have higher affinities for those epitopes than for the epitopes of the pathogens on the right, which are not complementary.

Such activation thresholds allow lymphocytes to function as *generalized* detectors: a single lymphocyte can detect (be activated by) structurally similar kinds of epitopes. If we consider the space of all epitope structures as a set of patterns, then a lymphocyte detects or "covers" a small subset of these patterns (called a similarity subset). Hence, there does not have to be a different lymphocyte for every epitope pattern to cover the space of all possible epitope patterns. There is evidence to suggest that certain kinds of lymphocytes (memory cells) have lower activation thresholds than other lymphocytes, and so need to bind fewer receptors to become activated.

2.1.2 Receptor Diversity

Because detection is carried out by binding with nonself[5], the immune system must have a sufficient diversity of lymphocyte receptors to ensure that at least some lymphocytes can bind to any given pathogen. Generating a sufficiently diverse repertoire of receptors is a problem, because the human body does not manufacture as many proteins as there are possible pathogen epitopes. [Inman, 1978] has estimated that the IS has available about 10^6 different proteins, and that there are potentially 10^{16} different proteins or patterns to be recognized. One of the main mechanisms for producing the required diversity is a pseudo-random process, in which recombination of DNA results in different lymphocyte genes, and hence different receptors[6].

[Tonegawa, 1983] has estimated that there are at most 10^8 different receptors. If we assume that there are 10^{16} different pathogen epitopes, then there will not be a sufficient diversity of lymphocyte receptors to bind every single possible pathogen. We can expect replicating pathogens to evolve so that they cannot be detected. One solution of the immune system is to make protection dynamic, by providing a continual turnover of lymphocytes: each day approximately 10^7 new lymphocytes are generated [Osmond, 1993]. Assuming that there are at any given time 10^8 different lymphocytes, and these are turned over at a rate of 10^7 per day, it will take 10 days to generate a completely new lymphocyte repertoire. Over time, this turnover of lymphocytes (together with immune memory; see section 2.1.3) increases the protection offered by the IS.

2.1.3 Adaptation

The IS needs to be able to detect and eliminate pathogens as quickly as possible, because pathogens can replicate exponentially. The more generalized a lymphocyte is, the slower it will be at detecting specific pathogens, and the less efficient at eliminating them, so the IS incorporates mechanisms that enable lymphocytes to "learn" or adapt to the structures of specific foreign proteins, and to "remember" these structures for speeding up future responses. Both of these principles are implemented by a class of lymphocytes called B-cells, because they mature in the bone marrow.

When a B-cell is activated (i.e. its affinity threshold is exceeded), it produces copies of itself (clones that are produced through cell division). This copying process is subject to mutation rates that are nine orders of magnitude higher than ordinary cell mutation rates, called somatic hypermutation, which can result in daughter B-cells that have different receptors from the parent, and hence different pathogen affinities. These new B-cells will also have the opportunity to bind to pathogens, and if such binding exceeds their affinity threshold, they will in turn clone themselves. The higher the affinity of a B-cell for the pathogens present, the more likely it is that the B-cell will clone. This results in a Darwinian process of variation and selection, called affinity maturation: B-cells compete for available pathogens, with the highest affinity B-cells being the "fittest", and hence replicating the most. Affinity maturation enables B-cells to adapt to the specific pathogens present, so that the most successful B-cells will have a higher affinity for those specific pathogens. The higher the affinity the more efficient the lymphocyte will be in capturing specific pathogens, and consequently the

[5]Recall from section 1.1 that the term nonself encompasses all pathogens, toxins, etc. which are foreign to the body.

[6]This is a simplification. For research concerning receptor diversity, see [Oprea & Forrest, 1999].

faster the infection will be eliminated. This is particularly important when the IS is fighting off a replicating pathogen: we have a race between pathogen reproduction and B-cell reproduction.

A successful immune response results in the proliferation of B-cells that have higher than average affinities for the foreign pathogens that caused the response. Retention of the information encoded in these B-cells constitutes the "memory" of the IS: if the same pathogens are encountered in future, the pre-adapted subpopulation of B-cells can provide a response that is more rapid than the original response. Our understanding of immune memory is problematic because B-cells typically live only a few days, and once an infection is eliminated, we do not know what stops the adapted subpopulation of B-cells from dying out. There are two theories that are currently dominant. According to one of the theories, the adapted cells become memory cells that live up to the lifetime of the organisms [MacKay, 1993]. The other theory postulates that the adapted B-cells are constantly re-stimulated by traces of nonself proteins that are retained in the body for years [Gray, 1992].

A consequence of learning and memory is that two types of immune response can be distinguished: a primary response to previously unencountered pathogens, and a secondary response to pathogens that have been encountered before. A primary response will typically take up to three weeks, during which B-cells are adapting to the specific pathogens, whereas a secondary response is much briefer (on the order of a few days) because the IS retains a memory of those specific pathogens (see figure 2.2). A secondary response is not only triggered by re-introduction of the same pathogens, but also by infection with new pathogens that are similar to previously seen pathogens; in computer science terms, immune memory is *associative* [Smith, *et al.*, 1998]. This feature underlies the concept of immunisation, where exposure to benign forms of a pathogen engenders a primary response and consequent memory of the pathogen enables the IS to mount a more rapid secondary response to similar but virulent forms of the same pathogen (see figure 2.3).

2.1.4 Tolerance

The picture described thus far has a fatal flaw: receptors that are randomly generated and subject to random changes from somatic hypermutation could bind to self and initiate autoimmune responses. An autoimmune response occurs when the IS attacks the body. Autoimmune responses are rare[7]; generally the immune system is tolerant of self, that is, it does not attack self. Tolerance is the responsibility of another class of lymphocytes, called T helper cells (Th-cells), because they mature in the thymus, and "help" the B-cells. Most self epitopes are expressed in the thymus (an organ located behind the breastbone) so during maturation Th-cells are exposed to most self epitopes. If an immature Th-cell is activated by binding self, it will be censored (i.e., it dies by programmed cell death) in a process called clonal deletion or negative selection. Th-cells that survive the maturation process and leave the thymus will be tolerant of most self epitopes. This is called Central Tolerization (CT) , because the immature Th-cells are tolerized in a single location (the thymus).

B-cells are tolerized in the bone-marrow, but this is not sufficient to prevent the development of autoreactive B-cells. During affinity maturation B-cells hypermutate, which can result in previously tolerant B-cells producing autoimmune clones. Affinity maturation occurs in the germinal centres of the lymph nodes, of which there are hundreds distributed throughout the body. To ensure tolerance across these distributed locations requires what can be termed peripheral or Distributed Tolerization (DT) . Th-cells provide this through a mechanism known as costimulation. To be activated, a B-cell must receive costimulation in the form of two disparate signals: signal I occurs when the number of pathogens binding to receptors exceeds the affinity threshold (as described in section 2.1.1), and signal II is provided by Th-cells. If a B-cell receives signal I in the absence of signal II it dies.

[7] At the most, five percent of adults in Europe and North America suffer from autoimmune disease [Steinman, 1993].

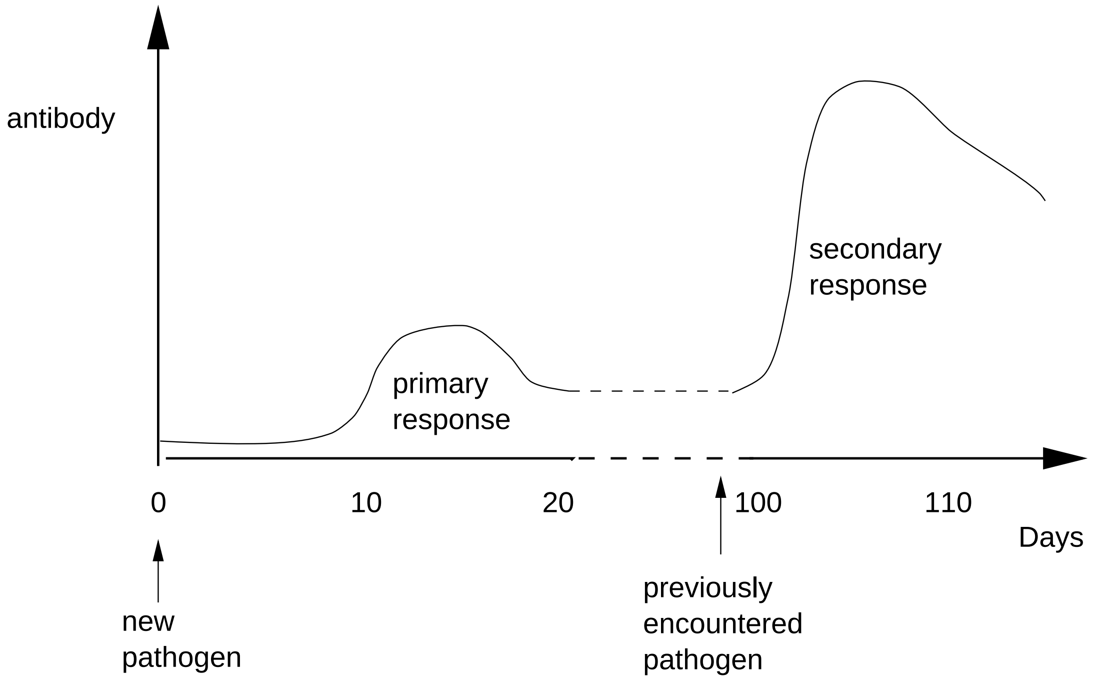

Figure 2.2: Responses in immune memory. Primary responses to new pathogen epitopes take on order of weeks; memory of previously seen pathogens epitopes allows the IS to mount much faster secondary responses (on the order of days). The y-axis (*antibody*) is a measure of the strength of the IS response.

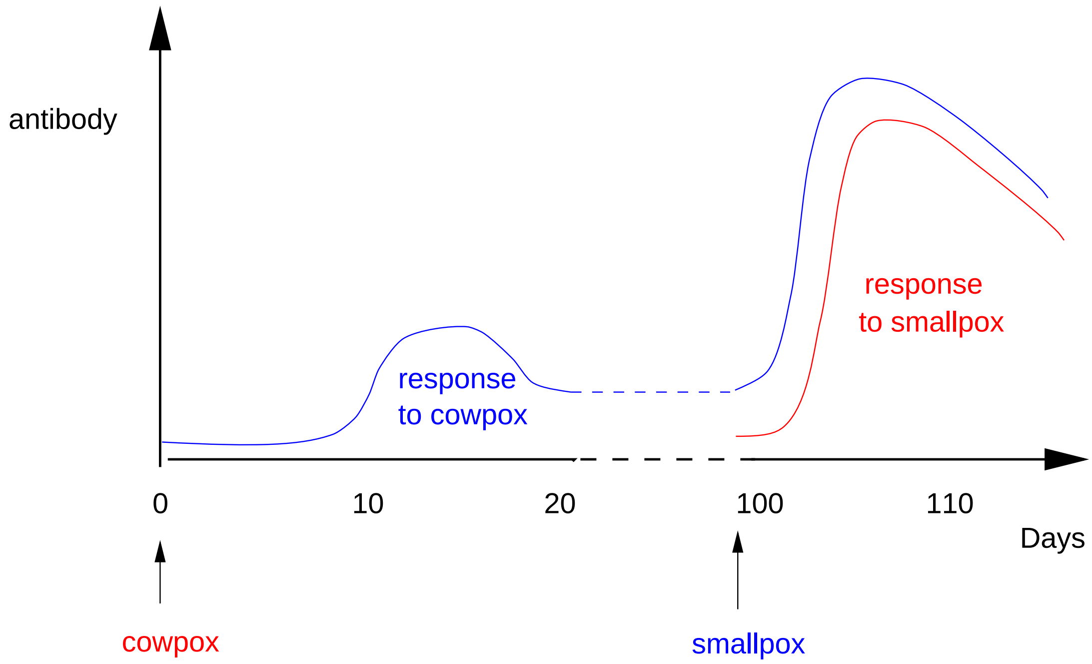

Figure 2.3: Associative memory underlies the concept of immunization. At time zero, the cowpox pathogen is introduced. Although harmless, it is recognized as foreign, so the IS mounts a primary response to it, clears the infection, and retains a memory of the cowpox. Smallpox is so similar to cowpox, that the memory population generated by the cowpox reacts to the smallpox, eliminating the smallpox in a more efficient secondary response.

To provide signal II to a B-cell, a Th-cell must "verify" the epitopes detected by the B-cell. The way in which it performs this verification is complex. B-cells engulf pathogen peptides and present these peptides on the surface of the B-cell, using molecules of the Major Histocompatibility Complex (MHC). These MHC molecules show the Th-cells what is inside the B-cell, that is, what the B-cell has detected. If a Th-cell binds to an MHC/peptide complex presented on the surface of a B-cell, it will provide signal II to that B-cell, and the B-cell will be activated. Because Th-cells undergo centralized tolerization in the thymus, most mature Th-cells are self-tolerant, and so will not costimulate B-cells that recognize self. The Th-cell "verifies" that the detection carried out by the B-cell is correct, and not autoimmune.

Unfortunately, the picture is not as simple as this. Some peripheral self proteins are never presented in the thymus (the exact fraction is unknown), and so Th-cells emerging from the thymus may still be self-reactive. Self-tolerance in Th-cells is also assured through costimulation: once again, signal I is provided by exceeding the affinity threshold, but signal II is provided by cells of the innate IS. These innate system cells are thought to give out signal II in the presence of tissue damage[8]. This may still allow autoreactive T-cells to survive, but as soon as an autoreactive Th-cell leaves the region of tissue damage, it will receive signal I in the absence of signal II and die. This can be termed frequency-based tolerization because an autoreactive Th-cell should encounter self in the absence of tissue damage with higher frequency than self in the presence of tissue damage, because healthy self is generally more frequent than nonself. If these frequencies change, then frequency tolerization can lead to a loss of immune function, which has been observed when overwhelming initial viral doses result in nonself becoming frequent [Moskophidis, *et al.*, 1993]. The utility of frequency tolerization is emphasized by the fact that the loss of the thymus does not result in devastating autoimmunity, which is what we would expect if Th-cells were only tolerized centrally.

The specialization of the different lymphocytes gives the immune system the ability to provide a faster adaptive response that is not self-reactive. Th-cells have the responsibility for self-tolerance, thus freeing B-cells to hypermutate and adapt to a specific pathogen. Although Th-cells must be able to recognize the peptides presented by the B-cells, both classes of lymphocytes are necessary. Th-cells are general, non-specific detectors, and so are not efficient at detecting specific pathogens. B-cells, by contrast, can adapt to become more specific, and thus more effective at detecting particular pathogens. It has been estimated that B-cells detect specific pathogens 10 to 10000 times more efficiently than Th-cells [Matzinger, 1994].

2.1.5 MHC and diversity

It is essential for host defense that MHC can form complexes with as many foreign peptides as possible, so that those foreign peptides can be recognized by Th-cells. Because of the nature of molecular bonding, a single type of MHC can form complexes with multiple, but not all pathogenic peptides. Hence, there is selective pressure on pathogens to evolve so that their characteristic peptides cannot be bound by MHC, because then they will be effectively hidden from the IS. Therefore, it is essential that the body have as many varieties of MHC as possible. However, as the diversity of MHC types increases, there is a resulting increase in the chance that immature Th-cells will bind to complexes of MHC and self, which means that more Th-cells will be eliminated during negative selection. Eliminated Th-cells are a waste of resources, so evolution should favour lower rates of Th-cell elimination during negative selection. Hence the number of MHC types is constrained from below by the requirement for diversity to detect pathogens, and from above by resource limitations imposed by negative selection. Mathematical models of this trade-off indicate that the number of MHC types present in the human body (about 4 to 8) is close to optimal [Mitchison, 1993].

MHC types do not change over the life of an organism and are determined by genes that are the most polymorphic in the body. Hence, MHC is representative of genetic immunological diversity within a popu-

[8] This is a simplification. For a more detailed exposition of possible tolerization and costimulation mechanisms, see [Marrack & Kappler, 1993, Matzinger, 1994, Matzinger, 1998].

lation. This diversity is crucial in improving the robustness of a population to a particular type of pathogen. For example, there are some viruses, such as the Epstein-Barr virus, that have evolved dominant peptides that cannot be bound by particular MHC types, leaving individuals who have those MHC types vulnerable to the disease [Janeway & Travers, 1996]. The genetic diversity conferred by MHC is so important that it has been proposed that the main reason for the continuance of sexual reproduction is to confer maximally-varied MHC types upon offspring [Hamilton, *et al.*, 1990]. There are some studies with mice that support this theory. These studies indicate that mice use smell to choose mates whose MHC differs the most from theirs [Potts, *et al.*, 1991].

2.2 A First Attempt at Applying Immunology to ID: Host-based Anomaly Detection

The first project the author was involved in that made use of immunology was that of designing and testing a host-based anomaly ID for the UNIX operating system [Hofmeyr, *et al.*, 1998, Forrest, *et al.*, 1997, Forrest, *et al.*, 1996]. Although this work was successful, it did not go far in incorporating ideas from immunology. The primary drive of the research was to define a "sense of self" for a computer system, that is, to find some characteristic of a computer system that was universal, robust to variations in normal behaviour, but perturbed by intrusions. In other words, we were looking for a computer system "peptide".

As our peptide, we chose short (of length six to ten) sequences of system calls emitted by running programs. We found these short sequences resulted in a compact signature of self (from 200 to 5000 unique sequences), and reliably discriminated between self and nonself. The method was simple, and was implemented in two phases: a training phase, and a test phase. During the training phase, a profile or database of normal behaviour was collected for each program of interest. The database was collected by scanning traces of normal behaviour and extracting sequences of system calls, ignoring parameters to the system calls and tracing forked subprocesses individually. Each normal database was specific to a particular system (defined by its configuration, architecture, software version, etc.). In the test phase, the running program was monitored for deviations from the recorded normal behaviour, that is, the system scanned for occurrences of sequences not in the normal database for that program. The method was deterministic: every sequence detected during testing that was not found in the normal database was flagged as a mismatch, and anomalies were determined by aggregating mismatches over a temporal window (typically 20 system calls).

Experimental results using this method indicate that self is compact (the normal databases were from 200 to 5000 unique sequences) and sequences of system calls distinguish many different types of anomalies. Normal databases of sequences differed by 42% to 84% between different programs, and there was clear detection of common intrusions (buffer overflows, symbolic link problems, trojan code and Denial of Service (DOS) [9]) on three different UNIX platforms, for nine different programs (sendmail, ftpd, lpr, swinstall, lpr, xlock, named, login, ps and inetd). Intrusions were detected with 1 to 20% mismatches counted over the duration of the intrusion trace. Furthermore, the method exhibited low false alarm rates, for example, monitoring lpr resulted in about 1 false alarm every other day.

Although inspired by thinking about immunology, this research incorporated little in the way of immune system algorithms and architecture. Three principles that were successfully incorporated were *implicit policy specification*, because acceptable behaviour was what the programs were observed to do during normal operation), *anomaly detection*, because the system was not trained on known intrusions, and *diversity*, because the local variations in program setup and usage between different systems resulted in different normal databases even for the same program. For example, we collected normal for lpr at both the Computer

[9] These intrusions are described in more detail in [Hofmeyr, *et al.*, 1998] and in [Warrender & Forrest, 1999].

Science Department at the University of New Mexico (UNM) , and at the Artificial Intelligence Laboratory at the Massachuesetts Institute of Technology (MIT) . It was found that these databases differed by 40% even though they were both databases of normal behaviour for the same program (lpr).

2.3 Network Intrusion Detection

The research on system calls was promising, but did not lend itself well to a distributed architecture: the ID system was host-based, and had nothing to gain from the distributed mechanisms used by the IS. By contrast, a computer network is a distributed environment, and for this reason, the domain of computer networks was chosen. Furthermore, monitoring network traffic has the advantage that all attacks on a system must come through the network, unless the attacker is sitting at the console of the target computer. Finally, computers communicate via standardized protocols, and so monitoring these protocols allows the ID system independence from the local operating system. This section gives a brief overview of some salient points in networking before moving onto a review of network intrusion detection, but the interested reader should consult [Comer, 1995, Hunt, 1992, Stevens, 1994].

2.3.1 Networking and Network Protocols

Information is sent across networks in discrete packets. There are two fundamental ways of transmitting these packets: broadcast and packet switched. On a broadcast network, all packets are sent to all computers on the network, and individual computers only accept the packets that are addressed to them (although they see all packets). This system works well for small networks of computers that are linked together by high-speed connections (10Mbit/s and more). Such networks are known as Local Area Networks (LANs) because the computers must be in physically proximity to one another to enable high-speed communication[10]. When computers are further apart, they are linked by Wide Area Networks (WANs) , and packets must be transmitted using packet-switching: packets have to be routed from source computer to destination computer via intermediate computers or routers, and the packets can take different routes, that is, they can be *switched* between different routes. The internet is an example of a packet-switched network that connects LANs together, which are predominantly broadcast networks.

Any attack launched against a computer over a network must make use of communications protocols. Monitoring network traffic at different protocol levels will yield different information about possible attacks. The basic protocol which links all computers on the internet together is the Internet Protocol (IP) . Other protocols are layered on top of IP.

Each computer communicating via IP is assigned a unique 32-bit address[11], which is written as four bytes, for example, 198.15.77.3. IP addresses are assigned by a central authority in blocks, for example, if $*$ is a don't care symbol, then a block of addresses could be 198.15.77.$*$, which would mean that the block contained 256 addresses. Each address block is referred to as a network, and is identified by its most significant bytes, so the block in the example above would be the network identified by 198.15.77. Blocks are allocated in various sizes; the size determines the network class. There are three different classes that are used for network addresses: class A networks are identified by only the most significant byte, so they contain 2^{24} addresses; class B networks are identified by the two most significant bytes, so they contain 2^{16} addresses; and class C networks are identified by the three most significant bytes, so they contain 256 addresses.

Three main protocols that are layered on top of IP are:

[10] Not all LANs use broadcast, an issue that is returned to later in section 7.3.

[11] Computers on different networks that are isolated from one another can have the same IP addresses.

Internet Control Message Protocol (ICMP) : This protocol is used for low level communication of control information between computers on the internet. It enables computers to check routes to other computers, to change routes to other computers, and similar functions.

User Datagram Protocol (UDP) : This is a connectionless protocol: packets are sent individually and there is no guarantee that the packets will get to their destination or that they will arrive in the correct order. For this reason, UDP is sometimes mistakenly believed to stand for *unreliable* datagram protocol. However, allowing for unreliability reduces the overhead and can make UDP faster than a connected protocol. Hence, UDP is useful for communications which are not adversely affected by the occasional loss or misplacement of a packet, or for communications where the speedup is important.

Transmission Control Protocol (TCP) : By contrast to UDP, TCP is a reliable, connected protocol. Information that is transmitted using TCP will always reach its destination in the order transmitted, or the communicating computers will be informed of a transmission error. A connection is established by a three-way handshake (see figure 2.4).

Because each computer can support multiple, simultaneous connections, each connection is identified by a 16 bit number, called a port. The ports on the two computers involved in a connection can be (and generally are) different. Hence, every connection can be identified by the 4-tuple (source IP address, destination IP address, source port number, destination port number). Certain ports are assigned to particular services: by convention, particular servers will respond to connection requests on particular ports. For example, a web server will respond to requests to connect to port 80, irrespective of the source port requesting the connection. Generally, only servers use assigned ports; clients use unassigned, higher-numbered ports. In the UNIX operating system, ports are further classified as privileged (those under 1024) or nonprivileged (from number 1024 upwards). Usually privileged ports are reserved for connections to privileged services, such as login, and regular users are not allowed to run programs that listen to these ports. TCP is used when reliable two-way communication is required, for example, with file transfers.

2.3.2 Network Attacks

Attacks can be divided into two classes: host-targeted attacks which exploit holes in programs running as servers, and network-targeted attacks that exploit weaknesses in network protocols. In the previous work described in section 2.2, we developed a method for detecting host-targeted attacks, but not network-targeted attacks. There are multiple varieties of network-targeted attacks, most of which rely on the fact that under the standard protocols, any packets that can be intercepted can also be read. Sensitive information (such as passwords) can then be used to compromise security. The best way to deal with this problem is to encrypt the traffic, so that even if an attacker can intercept the packets, the attacker cannot read them.

Whatever the nature of the attack, it must come over the network unless the attacker is physically present at the computer. In typical attack scenarios, the attacker does not know much about the target network of computers, and must probe the network in various ways to glean information so that an attack can be launched. The sophistication of this probing can vary. An example of unsophisticated, crude probing is portscanning, where the attacker attempts to connect to every single port on the target computer. Another example is blind address-space probing, where the attacker attempts to connect to every IP address in a network, regardless of whether or not a computer exists at that address. Generally, more sophisticated probing is more limited, for example, an attacker might only probe those ports that run services with vulnerabilities discovered within the last month. Another way in which an attacker can make probing more stealthy is to spread out the actions over a longer period of time, so there are no bursts of suspicious activity. The attacker

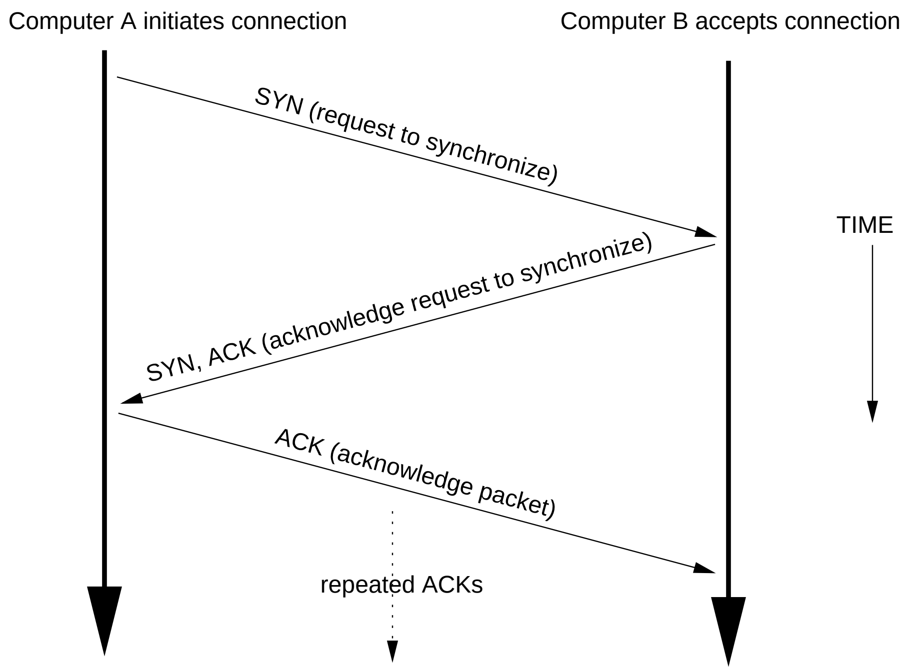

Figure 2.4: The three-way TCP handshake for establishing a connection. The computer requesting the connection (A) sends a TCP packet to B, with the SYN flag of the packet set. SYN stands for *synchronize* and is a request for a connection. In accepting the connection, B sends a return packet to A with the SYN flag set, and the ACK flag set. ACK stands for *acknowledgment* and indicates that B has received the request for a connection and accepts it. From that moment on, information is exchanged using only the ACK bit, to acknowledge that previous packets were received.

could also spread out the attacks over IP space, by probing each port or address from a different location. Such a Distributed Coordinated Attack (DCA) can be much harder to detect.

2.3.3 A Survey of Network Intrusion Detection Systems

Network ID systems are designed to detect network attacks by analysing and monitoring network traffic[12]. Some good overviews of ID systems are given in [Mukherjee, *et al.*, 1994]. This section briefly describes existing network ID systems, including NADIR [Hochberg, *et al.*, 1993], DIDS [Snapp, *et al.*, 1991], EMERALD [Porras & Neumann, 1997, Porras & Valdes, 1998], AAFID [Crosbie & Spafford, 1994, Crosbie & Spafford, 1995a, Crosbie & Spafford, 1995b, Balasubramaniyan, *et al.*, 1998], NID [Heberlein, 1998], NSM [Heberlein, *et al.*, 1990, Heberlein, *et al.*, 1991, Mukherjee, *et al.*, 1994], GrIDS [Staniford-Chen, *et al.*, 1996], NetSTAT [Vigna & Kemmerer, 1998], NetRanger [NetRanger, 1999], and ASIM [ASIM, 1996][13]. Many of these systems have both network and host-based components (NADIR, DIDs, EMERALD, AAFID, GrIDS), but this discussion focuses on the network monitoring components.

Most of the network ID systems described here perform signature-based detection; few perform anomaly detection (NSM, EMERALD and to a limited extent NID and ASIM). Some systems have only a single monitor (NSM, NID, ASIM), but most use multiple monitors distributed across the network. These multi-monitor systems collate information in one of two ways: either centrally (NADIR, DIDs, NetSTAT, NetRanger) or hierarchically (EMERALD, AAFID, GrIDS). The hierarchical architectures have the advantage of scalability. The largest difference between the signature-based systems is in the way they model intrusion signatures. The most common approach is to use expert systems (NADIR, EMERALD, DIDs, NID). More unusual approaches include a method of encoding attack signatures with state transition diagrams (NetSTAT), and representing attack signatures as graphs (GrIDS). The three systems (NSM, EMERALD, ASIM) that perform anomaly detection both use some form of statistical analyses. In the case of AAFID, it is not clear what analysis methods are or will be used. The architecture is flexible enough to support different analysis methods; indeed, it is flexible enough to support both signature-based and anomaly detection.

None of these systems achieves all of the immunological principles laid out in section 1.3. There are some principles that are obviously achieved: all the systems except NSM perform signature-based detection and so have the equivalent of *memory*. Three systems (NSM, EMERALD, ASIM) perform *anomaly detection*, and these systems also have an *implicit policy specification*, because any anomaly detection system implicitly defines normal or common behaviour as legitimate. Some multi-monitor systems achieve *scalability* through hierarchical organizations.

Other principles are only achieved to a limited degree. The multi-monitor systems are *distributed*, in the sense that the monitors they employ are distributed throughout the network, and they can detect intrusive activity that happens over multiple computers. However, they are not localized in the sense that the IS is localized: they employ hierarchies, which means that they are vulnerable to attacks at high levels in the hierarchy. Multi-monitor systems are *robust* to the extent that loss of individual monitors will not result in a total failure of the system, however, as mentioned before, they are not robust to loss of, or faults in, components high in the hierarchy. The only ID system that shows promise of *diversity* is AAFID, because its architecture is general enough to employ a variety of detection methods simultaneously. The other systems do not achieve diversity because the same detection method is used within each system, and is replicated across all monitors.

There are two principles that all systems fail to achieve. None of the systems is *adaptable* in the way the IS, that is none of them can automatically extract intrusion signatures from novel attacks. The other

[12]Systems that monitor network traffic are commonly labeled “network-based” [Mukherjee, *et al.*, 1994].

[13]Because of the number of these systems, these acronyms are not expanded here; consult the glossary for the full names.

principle not achieved by any system is *flexibility*: in the context of the immune system, flexibility means being able to dynamically and automatically use resources as required, or as the opportunity offers. Note that AAFID has an architecture that could support both flexibility and adaptability, but there is no notion of automatic adaptation or reconfiguration in the current models.

2.3.4 Building on Network Security Monitor

The goal of this research was to build a prototype system that could achieve most of the immunological principles described above. A first requirement was anomaly detection. It would save time to build on some previous system that had already demonstrated success in anomaly detection, which meant that the only option was the Network Security Monitor (NSM) (AAFID can support anomaly detection but there is no description of what that anomaly detection could be, and no results of the efficacy of EMERALD's anomaly detection component have been reported). In this section, the concepts behind NSM are described in more detail. This material is drawn from [Heberlein, *et al.*, 1990].

NSM monitors TCP/IP traffic on a broadcast LAN, which has the advantage that NSM can see all packets on the LAN without requiring any additional auditing. The benefit of this passive monitoring is that using NSM does not place any overhead on the network. The TCP/IP traffic is represented as datapaths, where a datapath is the triple (source computer, destination computer, network service) representing a possible TCP connection between a source and destination computer using a particular network service (see figure 2.5). The normal traffic on the network can be characterised by a set of datapaths, and these can be used as the basis for an anomaly detection system.

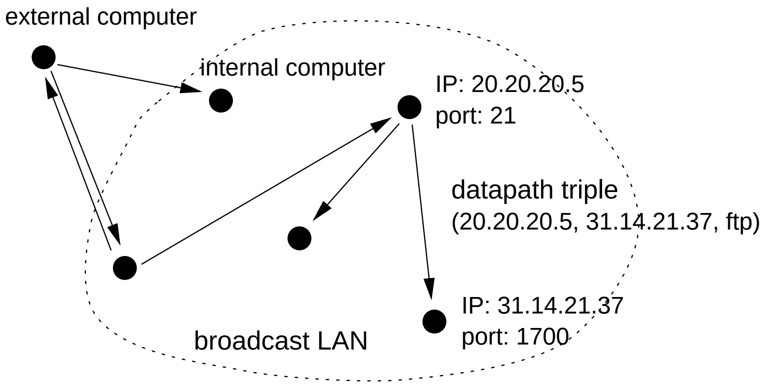

Figure 2.5: Patterns of network traffic on a broadcast LAN. Each computer can be thought of as a node in the graph, with a connection between computers forming an edge between the corresponding nodes. The dotted line indicates the boundary between the LAN and the external world. Note that a collection of normal datapaths will include paths between computers on the LAN, and connections between computers on the LAN and external computers.

NSM stores the profile of normal datapaths in a traffic matrix. Each cell in the matrix stores two statistics about a particular datapath: the frequency of connections on that datapath, and the rate of packet transfer over that datapath. In results reported in [Heberlein, *et al.*, 1990], the normal profile was collected over a period of two weeks, during which any connection that occurred at least once was regarded as normal (this includes connections between internal computers, and between internal computers and external computers). NSM monitors network traffic for anomalies whenever a new connection is added and at five minute intervals. If the new connection has a low probability of existing in the normal profile, or if traffic flow over an existing connection has a low probability of being at that frequency (where "low" is defined by an operator, according to security policy), then an anomaly is signaled, which may or may not indicate an attack. Carrying out such analysis exhaustively for all datapaths was found to be too computationally expensive, so analysis was performed on hierarchical decompositions of the traffic matrix[14].

In tests over two months at the University of California, Davis, NSM classified 300 out of 11000 connections as anomalous, and subsequent investigations revealed that most of these 300 anomalies were indicative of attacks. NSM was this successful because most computers in the LAN communicate with few (3 to 5) other computers, over few services. It was found that of all the possible datapaths in the LAN, only 0.6% were included in the normal, which means that an attacker had a low (0.06) probability of following normal paths and hence a high (0.94) probability of generating anomalous connections.

2.3.5 Desirable Extensions to NSM

NSM appears to be the right approach because it uses anomaly detection, normal is well-defined and stable and attackers are likely to perturb normal. To move closer towards the principles embodied in the immune system, several extensions to NSM are required, namely to make it distributed, robust, adaptable, and flexible. To achieve these, the concept behind NSM needs to be embodied in multiple detection systems that run on all computers on the LAN. These local detection systems must be sufficiently flexible and lightweight so that they can make full use of idle cycles on computers without compromising the usability of those computers.

Distributing NSM according to the immunological principles will make the system more resilient to insertion and evasion attacks [Ptacek & Newsham, 1998] against the ID system itself. An insertion attack is where an attacker constructs packets so that the ID system accepts them but the victim computer rejects them. The ID system could potentially be confused by such packets, especially if it is performing signature-based detection. An evasion attack is the reverse of an insertion attack: the attacker constructs a packet that is accepted by the victim computer, but not the ID system, which can be done by exploiting differences in the TCP/IP implementation between the victim computer and the one on which the ID system resides. Both of these kinds of attacks will be harder to execute if all computers on the network are running ID systems, including the victim computer. The attacker must now craft a packet that is rejected by a single computer but accepted by all others on the LAN, or vice versa.

2.4 An Immunologically-Inspired Distributed Detection System

This dissertation will show that the goals stated above, for an ID system based on NSM, can be achieved through the use of algorithms and architecture that are based on immunological mechanisms. Previous work has isolated and investigated an immunological model of change detection, called negative detection. [Forrest, *et al.*, 1994, D'haeseleer, *et al.*, 1996, D'haeseleer, 1996, D'haeseleer, 1995, Helman & Forrest, 1994]. This research is briefly reviewed here; more details are given later in chapter 3, in the formal description of the model of distributed detection which is a foundation for the results in this dissertation.

The problem of anomaly detection can be expressed as the problem of distinguishing *self* from *nonself*, where self is all legitimate and acceptable patterns, and nonself is all illegitimate or unacceptable patterns. These patterns could be a variety of characteristics of a system, for example, sequences of system

[14]A later version of NSM used an expert system to look for intrusion signatures in the traffic matrix, in addition to the statistical anomaly detection [Mukherjee, *et al.*, 1994]. NSM is still in use in this later form. In its current incarnation it is called NID, which stands for Network Intrusion Detector [Heberlein, 1998]. Another direct development of NSM is ASIM.

calls, or datapaths in a network. In this model, patterns are represented by strings of fixed length ℓ, most commonly binary strings. These strings are analogous to peptides in the immune system. The negative detection system works by learning the set of self strings in a training phase, and then classifying new data as normal or anomalous during a test phase.

A negative detection system consists of multiple negative detectors, each of which is represented by a string of length ℓ. These detectors are analogous to lymphocytes in the immune system, and detection between a detector d and a string s is analogous to receptor binding, which is modeled as partial string matching between s and the string representing d. In the work discussed here, the contiguous bits rule was used [Percus, *et al.*, 1993]: two strings match under the contiguous bits rule if they have the same symbols in at least r contiguous positions (r is the match threshold). The detectors are called *negative* because they are generated so that they match nonself strings, and not self strings. Hence, in a *negative* detection system, self is defined in terms of its complement.

The negative selection algorithm is used to generate the detectors, and is similar to negative selection (clonal deletion) in the thymus. During the training phase, candidate detectors are generated at random and compared to all self strings. If a candidate detector matches any self strings, it is deleted and replaced by a new randomly-generated detector, and the process is repeated. The result is a set of detectors that match nonself, and have the property that when a detector is activated, it is immediately obvious that a nonself string has been detected and its location can be pinpointed. This property means that the detectors can be distributed across multiple computers without requiring communication (memory sharing) between them.

The feasibility of negative detection was demonstrated for the problem of protecting files in the DOS operating system from corruption by viruses [Forrest, *et al.*, 1994]. The self set was derived from the files by dividing the files into equal-sized substrings of 32 bits in length, and the contiguous bits rule was used with a matching threshold of between 8 and 13 bits. Infections representing nonself were generated by file infector viruses. For example, one self set had 655 strings, which could be protected at 100% reliability by no more than 10 detectors. Similar results were obtained with boot-sector viruses.

Negative detection incorporates many of the immunological principles:

- Detection can be distributed by placing different detectors on different computers.
- Having different randomly-generated detectors on different computers confers diversity, because different computers will be able to detect different intrusions.
- The system is robust because the loss of some detectors on a single computer will not result in a complete absence of protection.
- If the self set is representative of empirical normal behaviour, then policy is implicitly specified.
- The system is flexible in that protection and computational requirements both depend on the number of detectors, so the number of detectors can be automatically increased to take advantage of idle cycles or decreased so as not to interfere with normal operation.
- Because detection is localized and does not require communication, the system is scalable.
- The system has the ability to do both anomaly and signature-based detection.

Chapter 3

An Immunological Model of Distributed Detection

The IS is a complex distributed system with a variety of novel mechanisms that can be copied to solve problems of distributed detection. To understand the issues involved, the problem of distributed detection is defined in an abstract framework that can be applied to different engineering problem domains. This framework can be used for analysing models of distributed detection; in particular, this dissertation is concerned with the issues of errors in detection, and what effects these have on the scalability and robustness of the system.

In this chapter, firstly, the framework is used to derive properties that are necessary for scalability and robustness (section 3.1). Next a model is described, one that is based on the IS, that exhibits these properties. This model uses negative detection, as presented in section 2.4. The implementation of the model is discussed, and analysis the properties of the model presented (section 3.2). The point behind this is to formalize the claim that the immunological model is an effective way of achieving the desirable principles listed in section 1.3. To this end I will mention analogies to the IS when appropriate.

3.1 Properties of The Model

The first goal is to define the problem that we are attempting to solve. The definition includes a notion of distribution, and some idea of resource limitations, both of which are essential for understanding the issues involved.

3.1.1 Problem Description

The model environment is defined over a universe U, where U is a finite set of finite patterns[1]. U is partitioned into two sets, S and N, called self and nonself, respectively, such that $S \cup N = U$ and $S \cap N = \emptyset$. Self patterns represent acceptable or legitimate events, and nonself patterns represent unacceptable or illegitimate events. Kolmogorov complexity is used to introduce the notion of resources. The Kolmogorov complexity[2], $\Gamma(U_i)$, of a set of patterns $U_i \subseteq U$ is the length (in bits) of the smallest program that outputs U_i [Chaitin, 1990, Kolmogorov, 1965].

[1] The term "pattern" is used to emphasise that this definition is not dependent on the representation of a pattern. In general, a "pattern" can be thought of as a finite length string; in the IS, a pattern is a peptide.

[2] Also called the Chaitin or descriptive complexity

The problem faced by a detection system, is: given limited resources, classify a pattern $s \in U$ as *normal*, corresponding to self, or *anomalous*, corresponding to nonself. A detection system $\mathcal{D}$ consists of two components, $\mathcal{D} = (f, \mathcal{M})$, where f is a binary classification function, and $\mathcal{M}$ is a set of patterns drawn from U representing the memory of the detection system, $\mathcal{M} \subset U$. To represent $\mathcal{M}$ will require $\Gamma(\mathcal{M})$ bits. The classification function f maps the memory set $\mathcal{M}$ and a pattern $s \in U$ to a binary classification of *normal* or *anomalous*, that is, $f : U^* \times U \to \{normal, anomalous\}$, where U^* is the power set of U. A pattern $s \in U$ is *normal* if it is in the memory, and is *anomalous* otherwise, that is,

$$f(\mathcal{M}, s) = \begin{cases} normal & \text{if } s \in \mathcal{M} \\ anomalous & \text{otherwise} \end{cases}$$

Classification errors are defined over a subset, U_{test} of U, called the test set, $U_{test} \subseteq U$. The test set can contain sets of both self and nonself strings, which are denoted N_{test} and S_{test}, respectively, $N_{test} \cup S_{test} = U_{test}$. A detection system, $\mathcal{D}$ can make two kinds of classification errors: a false positive error ε^+ occurs when a self pattern s is classified as *anomalous*, that is $\varepsilon^+ = s \in S_{test} \mid f(\mathcal{M}, s) = anomalous$; and a false negative error ε^- occurs when a nonself pattern s is classified as *normal*, that is $\varepsilon^- = s \in N_{test} \mid f(\mathcal{M}, s) = normal$. See figure 3.1. A detection event occurs when a pattern is classified as *anomalous*, regardless of whether or not the classification was in error. A detection system, $\mathcal{D}$, is said to *detect* a pattern $s \in U$ when $f(\mathcal{M}, s) = (anomalous)$.

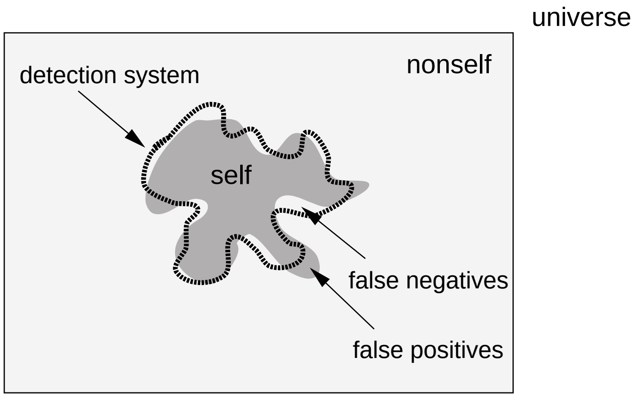

Figure 3.1: The universe of patterns. Each pattern can belong to one of two sets: self and nonself. Self patterns represent acceptable or legitimate events, and nonself patterns represent unacceptable, or illegitimate events. In this diagram, each point in the plane represents an pattern; if the point lies within the shaded area it is self, otherwise it is nonself. A detection system attempts to encode the boundary between the two sets by classifying patterns as either *normal* or *anomalous*. Where it fails to classify self patterns as *normal*, false positives are generated; nonself patterns that are not classified as *anomalous* generate false negatives.

3.1.2 Distributing the Detection System

The distributed environment is defined by a finite set L of locations, where the number of locations is n, that is, $|L| = n$. Each location $l \in L$ has a memory capacity that can contain at most M_l bits. In this distributed environment, each location $l \in L$ has a detection system $\mathcal{D}_l = (f_l, \mathcal{M}_l)$, where $\Gamma(\mathcal{M}_l) \leq M_l$. There are two separate, sequential phases of operation to the system: the first phase is called the training phase, and the second is called the test phase. During the training phase, each local detection system, $\mathcal{D}_l$, has access to a training set, U_{trn}, which can be used to initialize or modify the memory of $\mathcal{D}_l$. During the test phase, each detection system at each location l, attempts to classify the elements of an independent test set, $U_l \subseteq U$, with subsets $N_l \subset N$ and $S_l \subset S$, such that $N_l \cup S_l = U_l$. The performance of each detection system in terms of classification errors is measured during the test phase.

The immunological model that is the basis for this research does not use communication between local detection systems. There are advantages to be gained from avoiding communication. To understand why this is the case, a definition of communication is included in the framework, so that the effects of communication can be analysed. Communication is defined in terms of memory sharing. There are other definitions of communication possible; this one was chosen for simplicity. A detection system $\mathcal{D}_l$ at location l has access to the information contained in all other locations that it communicates with, so if $\mathcal{D}_l$ *communicates* with a set of locations $L_c \subseteq L$, this means that the classification function f_l for $\mathcal{D}_l$ is redefined as:

$$f_l(\mathcal{M}_l, s) = \begin{cases} normal & \text{if } s \in \bigcup_{j \in L_c} \mathcal{M}_j \\ anomalous & \text{otherwise} \end{cases}$$

It is assumed that if two locations communicate, then this costs a constant amount, $c \geq 1$. It is further assumed that all locations can communicate with any others. Note that there is no notion of time here; either a pair of locations communicate, sharing memory, at cost c, or they do not communicate.

Global properties (properties that are a consequence of considering all locations in combination) are defined to describe the overall performance of the system. The global classification function g of the distributed system is defined as

$$g(\{\mathcal{M}_l\}, s) = \begin{cases} anomalous & \text{if } \exists l \in L \mid s \in U_l \text{ and } f_l(\mathcal{M}_l, s) = anomalous \\ normal & \text{otherwise} \end{cases}$$

So, without communication (memory sharing), a pattern s will be classified globally as *anomalous* if at least one detection system has classified it as *anomalous*; conversely, if every location communicated with every other location, then s would be classified globally as *anomalous* only if all detection systems classified it as *anomalous*, because having all locations communicate is the same as having a single detection system with memory $\bigcup_{l \in L} \mathcal{M}_l$.

Global errors are defined as follows. An pattern $s \in S$ is a global false positive, ε_g^+, if it occurred in some training set for some detection system, and was classified as *anomalous*, $\exists l \in L \mid s \in S_l$ and $f_l(\mathcal{M}_l, s) = anomalous \Rightarrow \varepsilon_g^+ = s$. Conversely, if $s \in N$, then s is a global false negative, ε_g^-, if, for every training set in which it occurred, the local detection system failed to classify it as *anomalous*, $\forall l \in L \mid s \in N_l, f_l(\mathcal{M}_l, s) = normal \Rightarrow \varepsilon_g^- = s$.

3.1.3 Assumptions

Seven assumptions are made concerning the system and the problem. These assumptions are necessary to demonstrate the conditions required to attain the properties of scalability and robustness, which are formally defined later. All of the assumptions are justified below. Included are assumptions made in the definition of the problem; they are restated here for clarity (assumptions 1, 2), and are stated first. It is assumed that:

1. U is closed and finite. For any given problem domain, patterns must be represented in some fashion. In this dissertation, a fixed size representation is used, and any fixed size representation implies a finite and closed universe.

2. $S \cup N = U$ and $S \cap N = \emptyset$. There may be cases in which this assumption does not hold, which means that there will be patterns that are both self and nonself. It will be impossible for *any* detection system

to correctly classify such *ambiguous* patterns, and so they will always cause errors. The analysis will be valid if there is a subset of U for which the partition applies.

3. $M_l > \Gamma(\{s\}), \forall l \in L, \forall s \in U$, that is, every location has sufficient memory capacity to encode or represent any pattern drawn from U. Any location that has insufficient memory capacity to encode even a single pattern would be useless, and can be disregarded. If there is a subset of locations for which this assumption holds, then the analysis applies to those locations.

4. $\forall l \in L, s \in U_l \Rightarrow \exists j \in L, j \neq l \mid s \in U_j$, that is, every pattern occurs in at least two different locations. Distribution only has an effect on the analysis if there is some commonality of patterns across locations. If each location was always presented with a set of completely different patterns from any other, then the analysis would devolve to a set of separate problems, one for each location, each with its own distinct universe, and distribution could never make any difference to the analysis.

5. It is assumed that $U_{trn} = S$, that is, the training set is the self set. In the application presented in this dissertation (chapter 4), this assumption does not hold. This issue is addressed later, in section 3.2.7.

6. $\{s \in N_l \mid f_l(\mathcal{M}_l, s) = normal\} \subset N_l, \forall l \in L$, that is, every location contributes to the detection of nonself (i.e., every local detection system correctly detects at least one nonself string in its local test set). If a local detection system is not contributing, then it is useless and it can be excluded from the analysis with no effect. Therefore, even if this assumption is relaxed, the analysis still applies to the subset of local detection systems for which this is true.

7. $|U_l| > |L|, \forall l \in L$, that is, the size of the test set at each location, is always greater than the number of locations. In the application presented in this dissertation, this is true.

3.1.4 Generalization

In real-world applications, detection systems must be able to encode information compactly because of resource limitations. This concept is formalized in the notion of generalized detection. A set G is an m-generalization of a set A, if and only if, $A \subseteq G$ and $\Gamma(G) \leq m$. If m is the maximum bits required to encode any pattern $s \in U$, $m = \max_{s \in U}(\Gamma(\{s\}))$, then for any subset U_i of U, there exists a m-generalization, that is, $\forall U_i \subset U, \exists G_i$ such that $U_i \subseteq G_i$ and $\Gamma(G_i) \leq m$.

This can be shown as follows. Given $U_i \subset U$, if $\Gamma(U_i) \leq m$ then the m-generalization is $G = U_i$. However, if $\Gamma(U_i) \geq m$, then G is created from elements *not* in U_i as follows. Construct the set B by drawing patterns from U_i^C (the complement of U_i) until the limit m is reached, that is, $B = \{s \in U_i^C\}$ such that $\Gamma(B) \leq m$. Then the m-generalization is $G = U - B$, because, given that U is closed (assumption 1), $U_i \subseteq U - B$, and $\Gamma(G) = \Gamma(B^C) = \Gamma(B) \leq m$. The construction of B is only possible if m is large enough to encode even the most complex pattern in U.

3.1.5 Scalable Distributed Detection

The notion of scalability is essential to distributed systems. For a distributed application to be general requires that it feasible for systems of a variety of sizes (numbers of elements). There are multiple ways of defining scalability; the definition used here assumes that the size of the detection problem is fixed[3]. The question this definition addresses is: for a fixed detection problem size (i.e. fixed universe), does the detection system

[3]For example, one way of defining scalability is in terms of problem size; how does the computational complexity scale with the size of the problem?

"scale" when it is distributed across an increasing number of locations? More formally, distributed detection is scalable if an increase in the number of locations from n to n', $n < n'$ does not violate two conditions:

1. There is no increase in the number of global errors, that is, if E is the number of global errors over n locations, and E' is the number of global errors over n' locations, then $E' \leq E$.
2. Communication costs do not increase more than linearly, that is, if the cost of communication with n locations is Cn, then the cost with n' locations is Cn', where C is a constant.

Computational time complexity and space complexity have not been included in this definition. Within the framework there is no notion of computational time complexity, and space complexity is defined by the memory capacity of each location, which is regarded as fixed, that is, the memory capacity cannot be increased to accommodate more complex detection systems.

If we want scalable distributed detection under the assumptions given in section 3.1.3, then false positives cannot be allowed, but false negatives can be allowed. Let the set of locations be L, with $|L| = n$ and add a new location $l' \notin L$. For each $s \in U_{l'}$, there exists an $l \in L$ such that $s \in U_l$, because of assumption 4. There are two cases:

s is nonself: that is, $s \in N_{l'}$. There are two possibilities: either s is already a global false negative, or is globally classified as *anomalous*. In both these cases, if the new detection system incorrectly classifies s, that is, $f_{l'}(\mathcal{M}_{l'}, s) = normal$, it will make no difference to the global false negative errors. However, if the new detection system correctly classifies s, and s was previously a global false negative, then the number of global false negatives is *reduced* by the addition of l'. This illustrates an important property of false negatives: as the number of locations increases, the number of global false negatives will not increase, but can decrease.

s is self: that is, $s \in S_{l'}$. There are two possibilities: either s is already a global false positive (i.e. globally classified as *anomalous*), or s is correctly classified as *normal*. In the former case, an incorrect classification of s by the new location cannot increase global errors. In the latter case, however, if $f_{l'}(\mathcal{M}_{l'}, s) = anomalous$, then, without communication (shared memory), s will be a new false positive and the number of global errors will have increased. This increase can be prevented by communication between l' and a location $l \in L$, that correctly classifies s, that is, $f_l(\mathcal{M}_l, s) = normal$). The communication means that the detection system at l' has access to the memory at l and so can classify s correctly.

However, consider the case where $f_{l'}(\mathcal{M}_{l'}, s_i) = anomalous$, for $i = 1, 2, \ldots, n$, where $s_i \in S_{l'}$, which is possible because of assumption 7, and where every s_i is presented at a different location $l \in L$ and no other, and is classified as *normal*, that is, $s_i \in S_i$, and $f_i(\mathcal{M}_i, s_i) = normal$, and $s_i \notin S_l, \forall l \neq i, \forall i = 1, 2, \ldots, n$. Then $\mathcal{D}_{l'}$ must communicate with every $l \in L$ to avoid increasing the number of global false positives, because for every s_i, the new detection system at l' must communicate with a *different* location. If each new location has to communicate with all previous locations, then communication costs do not increase linearly.

3.1.6 Robust Distributed Detection

This section defines the conditions that are required for a distributed detection system to be robust. Robustness means that the loss of functioning at a few locations will not cripple the system. There are varying degrees of robustness, so robustness is defined as follows: a system is *k-robust* if removal of a subset of locations $L_{\text{cut}} \subset L$, of size $k = |L_{\text{cut}}|$ does not result in detection system failure. Here detection system failure is defined as the occurrence of *any* global false positives (because of scalability), or complete failure to globally detect nonself. So, detection is k-robust if, $\forall L_i$ such that $L_i \subseteq L$ and $|L_i| \leq k$ then $\bigcup_{l \in L-L_i}\{s \in S_l \mid f_l(\mathcal{M}_l, s) = anomalous\} = \emptyset$, and $\bigcap_{l \in L_i - L_i}\{s \in N_l \mid f_l(\mathcal{M}_l, s) = normal\} \subset N_l$. Note that this definition includes removal of the empty set, so to be k-robust for any $k > 0$, requires no false positives.

It is shown that a scalable distributed detection where detection memories are all M_l-generalizations of the self set S will always be more robust than a system in which some memories are not M_l-generalizations of S, under the assumptions given in section 3.1.3.

First it is shown that any distributed detection system with memories that are all M_l-generalizations of the self set is $(|L|-1)$-robust. Because of assumption 5, a complete description of the self set is available. In general, it cannot be assumed that $\Gamma(S) < M_l$ or $\Gamma(N) < M_l, \forall l \in L$, consequently each memory $\mathcal{M}_l$ must be some M_l-generalization of some subset of U, which is possible because of assumption 3. Let every memory set $\mathcal{M}_l$ be a M_l-generalization of S, that is, $S \subset \mathcal{M}_l$ and $\Gamma(\mathcal{M}_l) \leq M_l, \forall l \in L$. Then the false positive rate is zero because $\forall l \in L$, $S \subset \mathcal{M}_l \Rightarrow f_l(\mathcal{M}_l, s) = normal, \forall s \in S$. If a subset L_i is removed from L, there still cannot be false positives, because each individual memory is a M_l-generalization of S. Furthermore, removing $L_i \subset L$ will only result in a gradual degradation in detection (i.e., a gradual increase in errors), because of assumption 6, that is, $\bigcap_{l \in L-L_i}\{s \in N_l \mid f_l(\mathcal{M}_l, s) = normal\} \subset \bigcup_{l \in L-L_i} N_l$, providing $L_i \neq L$. So removal of any subset $L_i < |L|$ will not cause false positives or catastrophic failure, which means that this system is $(|L|-1)$-robust.

Every detection system $\mathcal{D}_l$ with memory $\mathcal{M}_l$ which is not a M_l-generalization of S requires communication, for without memory sharing f_l operating on $\mathcal{M}_l$ can produce false positives, because $\exists s \in S \mid s \notin \mathcal{M}_l$. The robustness of a communicating system depends on how many local memories are required to form a M_l-generalization of S. If k memories are required, $2 \leq k \leq C \leq |L|$, (where C is the communication group size constant), and we assume that any set of k memories will form a (kM_l)-generalization of S, then the communicating system is $(C-k)$-robust, because as locations are removed, the remaining locations can always communicate until only $k-1$ locations are left, at which point f_l operating on any of the remaining $k-1$ memories can result in false positives. As the number of memories required to form a generalization decreases, so the robustness increases. At the limit, if $k = 2$ (only two memories are required), and $C = |L|$, then the system is $(|L|-2)$-robust.

False positives in a communicating system can be avoided altogether by modifying each classification function f_l so that it classifies patterns as *anomalous* only if there are enough memories (k) to form a (kM_l)-generalization of S, that is,

$$f_l(\mathcal{M}_l, s) = \begin{cases} normal & \text{if } s \in \bigcup_{j \in L} \mathcal{M}_j \text{ or } C \geq k \\ anomalous & \text{otherwise} \end{cases}$$

With this modified function, if the number of memories available is less than k, the number needed to form a M_l-generalization of S, f_l will always return $normal$. Hence, there can be no false positives, but there also cannot be classifications of *anomalous* when more than k locations have been removed. So there is failure of detection, in that nonself can never be detected. Hence, this solution is also $(C-k)$-robust.

In conclusion, any system for which not all memories are M_l-generalizations of S will require communication. Communication makes a system $(C-k)$-robust, where $2 \leq k \leq C \leq |L|$. A system for which all memories are M_l-generalizations of S is $(|L|-1)$-robust and hence is always more robust than a system in which not all memories are M_l-generalizations of S. As k decreases in size, the difference in robustness decreases, but then the amount of shared memory also decreases.

3.2 Implementation and Analysis

To achieve two of the principles listed in section 1.3, namely scalability and robustness, requires a distributed detection system where each local system uses a generalization of the self set, and where communication is minimized. The IS provides inspiration for the design of such a system. This section describes the implementation of a distributed detection system, one that is based on the architecture of the IS. This implementation is an extension of previous work [Forrest, *et al.*, 1994, D'haeseleer, *et al.*, 1996, D'haeseleer, 1996, Helman & Forrest, 1994] that was described in section 2.4. The salient points are reiterated here.

Firstly, every pattern must be represented in some way, and there must be some way of compactly encoding generalizations of patterns. All patterns $s \in U$ are represented by binary strings of length ℓ. A representation Λ is a function mapping a pattern in U to a string of length ℓ in U_R, that is, $\Lambda : U \rightarrow U_R$[4]. The size of U_R is then 2^ℓ. Note that because U_R is a representation of U, it could be that $|U_R| < |U|$. S will then be represented by the set of strings S_R and N will be represented by the set of strings N_R. Under the representation, $N_R \cap S_R = \emptyset$ and $N_R \cup S_R = U_R$, will hold if Λ is a one-to-one mapping and assumption 2 holds. In all the analysis presented here (except in section 3.2.7) it is assumed that the training set is the same as the representation of the self set (i.e. assumption 5 holds), $U_{trn} = S_R$.

Generalization is implemented using detectors and partial string matching. There are other ways of implementing generalized detection, for example, neural nets, but partial string matching is used because it was used in the previous immunological models that are the foundation for this work. A single detector is an abstraction of a lymphocyte in the IS, and receptor binding is modeled by partial string matching. A detector d is a binary string, so the Kolmogorov complexity of d is $\Gamma(\{d\}) \leq \ell$, because each detector is a binary string of length ℓ. Each detector d compactly represents a set of strings C_d, called the cover of d, which is determined by a match function, or match rule h, $h : U_R \times U_R \rightarrow \{match, nomatch\}$. The event that a matches b under h is denoted as $Match_h(a, b)$, and the event that a does *not* match b under h is denoted as $\neg Match_h(a, b)$, thus, $h(a, b) = match \Rightarrow Match_h(a, b)$ and $h(a, b) = nomatch \Rightarrow \neg Match_h(a, b)$. When there is no ambiguity, the match function will be omitted. All analysis here assumes that the match rules are both reflexive (i.e., $Match_h(a, a), \forall a \in U$) and symmetric (i.e., $h(a, b) = h(b, a), \forall a, b \in U$).

The cover of d is defined as $C_d = \{s \in U_R \mid Match_h(d, s)\}$, that is, the cover[5] is the set of all strings that are matched by d under the given match rule h. A detector d, together with a match rule h is a $(k\ell)$-generalization of any set $U' \subset U_R$, if $U' \subseteq C_d$, because $\Gamma(C_d) \leq k\ell$, where k is constant representing the bits required to encode the operation of the match rule. According to assumption 3, a location $l \in L$ always has enough memory capacity to encode any event $s \in U$, that is, $\Gamma(\{s\}) \leq M_l$. This assumption is modified as follows: any location l has enough memory capacity to encode any string $s \in S_R$, and store a match rule, thus, $k\ell \leq M_l, \forall l \in L$. Any detector d always has a Kolmogorov complexity of $\Gamma(\{d\}) \leq k\ell$. For brevity, a detector d is sometimes referred to as a generalization of set A, which means that d together with some match rule h forms a $(k\ell)$-generalization of A.

This section describes and analyses two different kinds of match rules, how detectors can be constructed so that they are generalizations of S_R, how different representations are useful for minimizing errors, and mechanisms for reducing false positives in the case when the training set does not include all of S_R.

[4]The representation could also be a variable length string of some alphabet other than binary; fixed length strings were used to ease analysis and implementation, and a binary alphabet was used because this gives the most flexibility for partial matching, an issue which is explained later.

[5]Note that any arbitrary string has a cover, because a detector is a string, and because it is assumed that matching is symmetric and reflexive.

3.2.1 Match Rules

The two match rules discussed here both have a parameter $0 \leq r \leq \ell$, which is a match threshold; by adjusting the value of r, the size of the cover of a detector d can be modified. In all cases, if $r = 0$, then the cover of d is all strings, $C_d = U_R$, and if $r = \ell$, then the cover is a single string, d, $C_d = \{d\}$. The lower the value of r, the more general the match; the higher the value of r, the more specific the match. Furthermore, note that both these rules are symmetrical and reflexive.

The first match rule considered is based on Hamming distance and is termed the Hamming match: two strings a and b match under the Hamming match rule if they have the same bits in at least r positions. So if $a = 011001$ and $b = 001111$, then only if $r \leq 3$ is there a match, that is, $Match_h(a, b)$. The probability[6] of two random strings a, b matching under the Hamming rule is

$$P(Match_{hamm}(a, b)) = p_{hamm} = 2^{-\ell} \sum_{i=r}^{\ell} \binom{\ell}{i}$$

This probability is derived by noting that $2^{-\ell}$ is the probability of a single string occurring, and $\binom{\ell}{i}$ is the number of strings in U_R that have the same bits in i positions.

The second match rule is called the contiguous bits rule, and has been used as a plausible abstraction of receptor binding in the immune system [Percus, *et al.*, 1993]. Two strings, a and b match under the contiguous bits rule if a and b have the same bits in at least r contiguous locations (see figure 3.2). The probability of two random strings a, b matching under the contiguous bits rule is [Percus, *et al.*, 1993]:

$$P(Match_{contg}(a, b)) = p_{contg} \approx 2^{-r} \left(\frac{\ell - r}{2} + 1 \right) \tag{3.1}$$

r = 4

0110100101   0100110100

1110111101   1110111101

Match   No match

Figure 3.2: Matching under the contiguous bits match rule. In this example, the detector matches for $r = 3$, but not for $r = 4$.

These rules have different matching probabilities for the same value of r; the way the matching probability changes with r also varies. For the Hamming rule, a decrease of 1 in r will change the match probability by

$$\Delta p_{hamm} = 2^{-\ell} \binom{\ell}{r - 1}$$

This means that the further r is from the middle value, the less the match probability will change with changes in r. For the contiguous bits rule, a decrease of 1 in r will change the match probability by

$$\Delta p_{contg} = 2^{-r} \left(\frac{\ell - r}{2} + 2 \right)$$

[6]The probability of an event A is denoted by $P(A)$.

which means that a decrease of 1 in r approximately doubles the match probability, for all values of r.

The choice of rule depends on the application, and on the representation (see section 3.2.6). In chapter 4 these two rules are compared in the context of network intrusion detection.

3.2.2 Detector Generation

In section 3.1 it was shown that for scalable, robust, distributed detection, it is required that the memory, $\mathcal{M}_l$, for every local detection system must be a M_l-generalization of the self set. Detectors are M_l-generalizations of some sets; it is necessary to ensure that they are also M_l-generalizations of the *self* set. One method is to ensure that the cover C_d of a detector d is a M_l-generalization of S_R, which means that it must include every $s \in S_R$, that is, $S_R \subseteq C_d$. Because C_d is the set of all strings that are matched by d, it is required that d match all strings $s \in S_R$.

An alternative method is to copy the IS and define generalization in terms of the complement of C_d, which is possible because U is closed. If C_d is generated so that it includes no self, that is, $\forall s \in S_R, s \notin C_d$, then the complement of C_d is a M_l-generalization of S_R, because $C_d^C = U_R - C_d \Rightarrow S_R \subseteq C_d^C$. Then it is required that d match *no* strings in S_R. Such a detector is a generalization of S_R, and is called a negative detector [Forrest, *et al.*, 1994] because it is generated to match the complement of self, that is, to match nonself. A negative detector is analogous to a lymphocyte in the IS, because a lymphocyte is tolerized to bind only nonself peptides.

Negative detectors can be generated using the negative selection algorithm described in section 2.4. As described before, the negative selection algorithm is an abstraction of the negative selection of lymphocytes that happens in the thymus. The negative selection algorithm will guarantee that a detector d is a generalization of the *training* set, that is, $U_{trn} \subseteq C_d^C$ and $\Gamma(U_{trn}) \geq \Gamma(\{d\})$. If $S_R \subseteq U_{trn}$ then C_d^C is a M_l-generalization of the self set[7]. A detector d which is a generalization of S_R is called a valid or tolerized detector, and is denoted $V(d)$ (see figure 3.3). The process of generating a valid detector using the negative selection algorithm is termed tolerization (borrowing a term from immunology), because the algorithm generates detectors that are *tolerant* of the self set. To generate negative detectors using the negative selection algorithm requires access to the self set during training, but no prior knowledge about the kinds of nonself strings that could be encountered. Consequently, if any nonself strings are detected during the test phase, then the system is performing anomaly detection (one of the principles listed in section 1.3).

The time complexity of the negative selection algorithm is proportional to the number of times a candidate detector must be regenerated before it becomes valid. The number of retries can be probabilistically computed as follows. Let d be a candidate detector, and let F_i be the event $\neg Match(d, s), \forall s \in S_R$, that is, the event that d does not match any s in S_R. Then the probability that d is a valid detector is[8]:

$$P(V(d)) = P(F_1 F_2 \ldots F_{|S_R|}) \tag{3.2}$$

If it is assumed that the probability of a candidate detector d matching a self string s is independent of its probability of matching other self strings, and the probability of a match is denoted by $p_M = P(Match(d, s))$, then $P(F_i) = 1 - p_M, \forall i = 1, 2, \ldots, |S_R|$, and hence

$$\begin{aligned} P(V(d)) &= P(F_1)P(F_2)\ldots P(F_{|S_R|}) \\ &= (1 - p_M)^{|S_R|} \end{aligned} \tag{3.3}$$

[7] In section 3.2.7 the case where $S_R \not\subseteq U_{trn}$ is analysed.

[8] The intersection of the events A and B is denoted as AB.

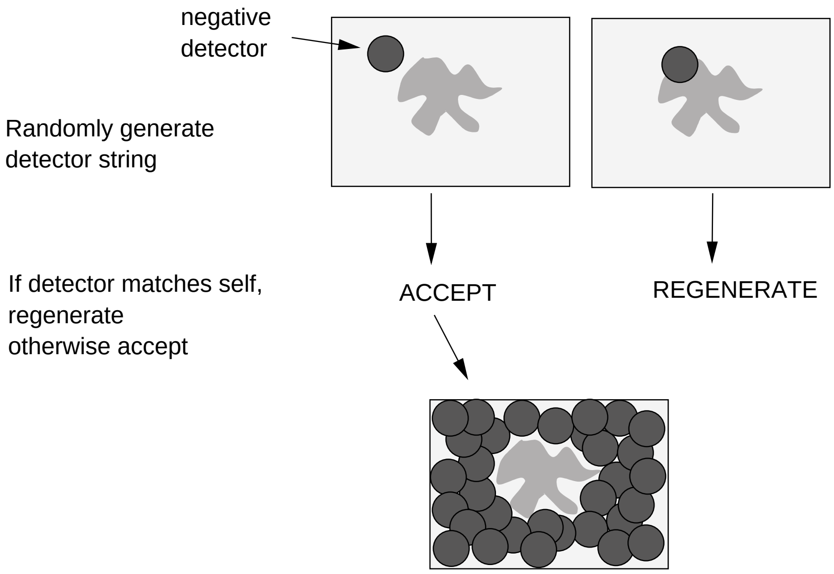

Figure 3.3: The negative selection algorithm. Candidate negative detectors (represented by dark circles) are generated randomly, and if they match any string in the self set (i.e. if any of the points covered by the detector are in the self set), they are eliminated and regenerated. This process is repeated until we have a set of valid negative detectors that do not match any self strings.

The number of retries required to generate a valid detector is a geometric random variable with parameter $P(V(d))$, so the expected number of trials[9], ρ until success is [Forrest, *et al.*, 1994]:

$$E(\rho) = \frac{1}{(1 - p_M)^{|S_R|}} \tag{3.4}$$

Hence the expected number of retries to generate a single valid detector is exponential in the size of the self set. This assumes that there are no similarities between self strings. The more similar the self strings, the fewer the number of retries. If this assumption does not hold, then equation 3.2 cannot be reduced to equation 3.3, because if a detector does not match one of the self strings, the probability of it matching the others is reduced. A more sophisticated analysis that takes into account similarities within the self set is given in section 3.2.4.

The negative selection algorithm can be applied to any match rule (and even to any detection operation). However, because of the exponential time complexity of negative selection, two other tolerization algorithms have been developed specifically for the contiguous bits rule: the linear time algorithm [Helman & Forrest, 1994], and the greedy algorithm [D'haeseleer, 1995]. Both algorithms trade space complexity for time complexity, that is, they have time complexities that are linear in the size of the self set, but space complexities on the order of $O((\ell - r)^2 \cdot 2^r)$, as compared to the space complexity of $O(|S_R|)$ for the negative selection algorithm. Although the time complexity is reduced, there are two reasons why these algorithms are not used in this dissertation. Firstly, the increased space complexity can be prohibitive for large enough values of ℓ and r (this is the case for the network ID application), and commonalities in the self

[9]The expectation of a random variable X is denoted as $E(X)$.

set do not reduce the space complexity, whereas commonalities in the self set reduce the time complexity for negative selection. Secondly, these algorithms are not general: they apply only to the contiguous bits rule, and so cannot be used for the Hamming match rule.

3.2.3 Detector Sets

The negative selection algorithm generates detectors that are valid. However, a given location may have sufficient memory to store more than one detector. This is analogous to the IS, in which a location such as a lymph node can contain multiple lymphocytes. Thus a local detection system $\mathcal{D}_l$ is implemented as a set D_l of η_l detectors, $|D_l| = \eta_l$, together with a match rule, h_l, that is, $\mathcal{D}_l = (D_l, h_l)$. A benefit of implementing local detection systems as collections of detectors is flexibility: detectors can be added and deleted as computational costs and memory allow.

In general, because the detectors are randomly generated, $\Gamma(D_l) \approx \eta_l k \ell$. If $S_R \subseteq U_{trn}$, $\mathcal{D}_l$ will not produce false positive errors. The matching probability, p_M, can be used to compute the relationship between the number of detectors and the probability of a false negative error, p_{ε^-}. Assuming that the detectors in D are drawn randomly from U, and the probability of one detector matching is independent of the other matching, then the probability that a random nonself string $s \in N_R$ is a false negative, ε^-, is [Forrest, *et al.*, 1994]

$$P(\varepsilon^-) = p_{\varepsilon^-} = (1 - p_M)^\eta \tag{3.5}$$

For example, if $\ell = 49$ and $r = 12$, then under the contiguous bits rule, $p_M = 0.005$, and a false negative probability of $p_{\varepsilon^-} = 0.01$ is achieved with $\eta = 920$ detectors[10]. This number of detectors is small compared to the $2^{49} = 6 \times 10^{14}$ possible strings, most of which could be nonself. Combining detectors in this way exhibits a useful property: the probability of an error decreases exponentially with an increase in the number of detectors. In the previous example, if the number of detectors is doubled to $\eta_l = 1840$, the error probability drops by two orders of magnitude to $p_\varepsilon^- = 0.0001$.

Equation 3.5 holds only if the probability of a random nonself string being matched by one detector is independent of the probability of matching by other detectors. This is approximately true if detectors are strings randomly drawn from U_R, and the self set is small in comparison to the universe, $S_R \ll U_R$, because then $N_R \approx U_R$ and hence the negative selection algorithm will not bias the detectors towards certain strings. Note that equation 3.5 is derived for the case where the nonself string is randomly drawn. If a local test set U_l contains a random subset of nonself strings, then this assumption is approximately true, but if the local test set contains a biased subset of nonself strings, then this result will not hold. This latter case is discussed in more detail in section 3.2.4.

A consequence of this implementation of generalization is that there is a computational trade-off between the number of detectors, η, in a set and the number of retries, ρ, needed to generate that set. As the match length r increases, the probability of a match p_M decreases (for both match rules) and hence the number of retries decreases, but there is a corresponding increase in the number of detectors required for a given false negative error probability. Combining equations 3.5 and 3.4

$$\eta = -|S| \frac{\log p_\varepsilon^-}{\log E(\rho)} \tag{3.6}$$

This trade-off is illustrated in figure 3.4.

[10] In this and other examples, the value of $\ell = 49$. This is not an arbitrary choice; it is the shortest bit string in which a single pattern for the network ID application can be encoded. See section 4.1.1.

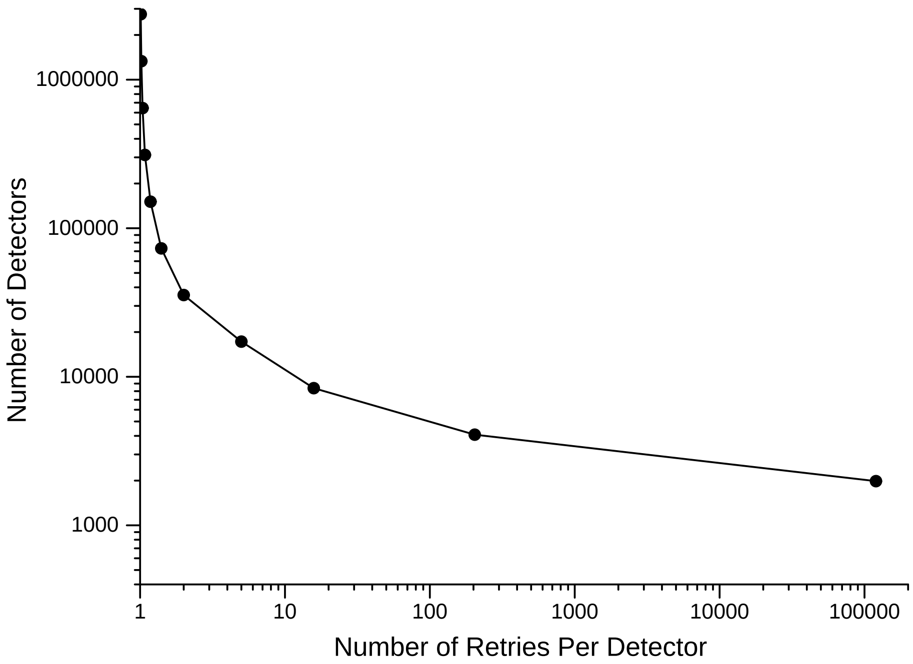

Figure 3.4: The trade-off between number of detectors required, η, and the expected number of retries $E(\rho)$. The curve is plotted from equation 3.6 by holding the false negative error probability fixed at $p_\varepsilon^- = 0.01$, the string length fixed at $\ell = 49$, and the size of the self set fixed at $|S_R| = 5000$. Not all points on the curve can be achieved, because r is an integer, so the points which can be achieved are marked by dots. Each dot on the curve corresponds to a different r value for the contiguous match rule, increasing from $r = 13$ on the right to $r = 23$ on the left. As the match length, r, increases, the number of retries decreases but the number of detectors required increases. Note that both axes are logarithmic.

3.2.4 The Existence of Holes

The assumptions in the analysis in the preceding section were that each local test set is randomly drawn from U_R and that the self set S_R is also a random set. If either of these assumptions fail, the analysis performed above will not apply. In reality, strings in the self set may not be randomly distributed, which means that they could be closer together in the match space. As the similarities in the set of self strings increases, so the number of retries required to generate a detector will decrease. However, as the similarities increase between the set of nonself strings (contained in a local test set) and the self strings, so the number of retries required to generate valid detectors will increase. For a match rule with a constant probability of matching, there can exist nonself strings for which no valid detectors can be generated. Such strings are called holes [D'haeseleer, *et al.*, 1996], because they are "holes" in the detection system's coverage of the nonself set. A nonself string $a \in N_R$ is a hole if and only if, $\forall s \in U_R$, such that $Match_h(s, a)$, s is a self string, that is, $s \in S_R$. See figure 3.5.

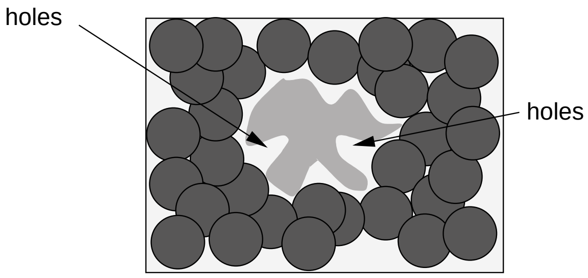

Figure 3.5: The existence of holes. There are patterns in the nonself set that cannot be covered by valid negative detectors of a given specificity. The size of the dark circles representing detectors is an indication of the generality of those detectors. The detectors depicted in the example here are too general to match certain nonself patterns without also matching self.

The concept of holes can be illustrated by an example. Consider three binary strings $s_1 = 001101$, $s_2 = 111111$ and $s_3 = 001111$. For the contiguous bits rule with $r = 3$, the subset of detectors that matches s_3 is $D_3 = S_{001***} \cup S_{*011**} \cup S_{**111*} \cup S_{***111}$, where S_{001***} is the set of 8 strings instantiated by the template $001 * **$ ("$*$" is a "don't care" symbol indicating either a 0 or 1). All the detectors in D_3 match either s_1 or s_2. Now if both s_1 and s_2 are part of the self set, $s_1, s_2 \in S_R$, and s_3 is a nonself string, $s_3 \in N_R$, then any detector that matches s_3 will also match at least one string in the self set, and so a valid detector cannot be generated for s_3. Hence s_3 is a hole.

The number of holes is dependent on the generality of the match: as r decreases, so the number of holes increases. In the limit, if $r = 0$, every nonself string will be a hole and no detection will be possible. Alternatively, if $r = \ell$, then every detector can only match a single nonself string (itself) and there will be no holes. The number of holes not only depends on r and ℓ, but also on the similarities in the self set (the structure of the self set). As such, there is no general closed formula for the expected number of holes that is independent of the structure of the self set. An algorithm for computing the number of holes in the whole self set, when using the contiguous bits rule, is given in [D'haeseleer, 1995]. This algorithm has time and space complexities that are exponential in the match length (the time and space complexities are $O((\ell - r) \cdot 2^r) + O((\ell - r) \cdot |S_R|)$ and $O((\ell - r)^2 \cdot 2^r)$ respectively), and so can be infeasible to compute for longer match lengths on longer strings. Furthermore, the algorithm counts every single possible hole in the nonself set, even though the detection system is trying to detect only nonself strings that occur in each local test set.

An alternative is to compute whether a particular, given nonself string is a hole. Then a representative sample of the nonself strings in a local test set can be used to compute the probability of a nonself string being a hole. An algorithm has been developed that does this for the contiguous bits rule. Both the space and time complexities of this algorithm are linear in $\ell - r$, but these complexities are per nonself string. To count all the holes, the time complexity would be on the order of $O((\ell - r) \cdot 2^{\ell})$. The linear running time per nonself string is achieved by an approximation in the algorithm that can result in loss of accuracy: the algorithm does not count the exact number of holes but produces an upper bound on the number of holes. The algorithm has the advantage that no extra cost is required to determine when the approximation is being used, so we can identify the cases in which the result is approximate. In the network ID application, the algorithm is exact for $r > 10$, when $\ell = 49$.

The algorithm determines if an anomalous string a is a hole under the contiguous bits match rule for a given r. Define a string r-template as a string consisting of r contiguously defined positions, and $\ell - r$ undefined or "don't care" positions. For example, $**0110****$ is a 4-template over a string of length $\ell = 10$. A template is called the i-th template if the first defined position is the i-th bit; in the previous example, $**0110****$ would be the 3rd-template. A template a_r matches a string s if the r-contiguously defined positions in a_r occur at corresponding locations in s, and is denoted $Match(a_r, s)$. For example, $**0110****$ matches 1101100100, so $Match(**0110****, 1101100100)$. The algorithm is as follows:

1. Find a single r-template a_r that matches a but does not match any self strings, i.e. $Match(a_r, a)$ and $\neg Match(a_r, s), \forall\ s \in S_R$. If no such template exists, exit: a is a hole.

2. Attempt to construct a valid detector for a, using a_r as the matching template. This is done by searching the space of all strings matched by template a_r. At each iteration of the search, an additional specific bit a_i is added to the template, which is initially set to $a_i = 1$, and the resulting template $a_r a_i$ is checked against S_R to determine if it is valid, which is the case if $a_r a_i$ does not match any $s \in S_R$. If $a_r a_i$ is valid, the search continues. If $a_r a_i$ is invalid, a_i is set to the alternative bit, $a_i = 0$. The algorithm carries out a depth-first search of the string subspace defined by the r-template, backtracking when it reaches dead-ends. This means the algorithm is exponential in $\ell - r$, but it can be made tractable by limiting the search steps to a constant c, that is, to c nodes in the search tree. It is assumed that if a valid detector is not found in c steps, then such a detector does not exist. This assumption means that the algorithm gives an upper bound on the number of holes. The accuracy of this bound can be assessed by reporting the number of times the search is terminated by the bound c.

3. If a valid detector cannot be constructed from a_r, repeat steps 1 and 2 until either a valid detector is constructed, or none can be found, in which case a is a hole.

The algorithm can be illustrated by an example. Let $\ell = 10$, and $S_R = \{0100111100, 1000111111, 1100110100, 0010010011\}$. Is $a = 0010110100$ a hole when r = 3? The 3-template $a_r = **101*****$ matches a but does not match any $s \in S_R$, so the subspace defined by a_r can be searched. An attempt is made to construct a valid detector for a using a_r. The search is conducted in two parts: the left hand side of a_r and the right hand side. The search through the right hand side is shown in figure 3.6. The combined LHS and RHS searches yield 1110101010 as a valid detector.

It has been shown in [D'haeseleer, *et al.*, 1996] that holes can exist for all symmetrical match rules with constant probabilities. By definition, a nonself string $a \in N_R$ is a hole if and only if, $\forall s \in U_R$ such that $Match_h(s, a)$, we have $s \in S_R$. Thus a is a hole if every string that matches a is in the self set, because a valid detector must consist of a string that matches a, but if every such string is in the self set, then a valid detector for a does not exist. Because of symmetry, the set of strings that match a is simply the cover of a,

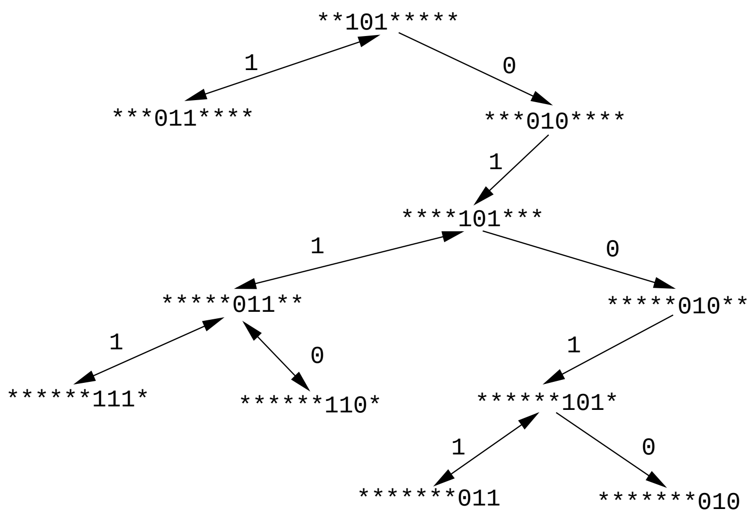

Figure 3.6: Searching the RHS string space for a valid detector. The search follows the arrows, always moving down and to the left first. The arrows are labeled with the defined bit value a_i added to a_r. Double ended arrows indicate backtracking. The 8-template $**10101010$ is valid.

that is, the set of strings *matched by* a. Then a self set of size $|S_R| = |C_a|$ is always sufficient to induce a hole. Here C_a is the cover of a, that is, all the strings matched by a. For a random self set, $|C_a| = p_M|U_R|$.

For some rules (such as the Hamming rule), the size of the self set necessary to induce a hole may be smaller. If $a \in N_R$ is a nonself string, and the bits of a are $a = a_1a_2 \ldots a_\ell$, then, for the Hamming rule with $r < \ell$, only one string is needed in the self set $\overline{a_1}a_2a_3 \ldots a_\ell$ (the over-line indicates the complement of the bit). For the contiguous bits rule two strings are needed: $\overline{a_1}a_2a_3 \ldots a_\ell$ and $a_1a_2a_3 \ldots \overline{a_\ell}$. Note that under the assumption that $N_R \cap S_R = \emptyset$, there can never be holes for completely specific match rules, that is, where $r = \ell$.

3.2.5 Refining the Analysis

A variant of the hole-counting algorithm described in section 3.2.4 is used to refine the predictions for the probability of a false negative error (equation 3.5), and the expected number of retries (equation 3.4), for the contiguous bits rule. Assuming detectors are independent of one another, and the same match rule is applied to all of them, then the probability of a detector set (at location $l \in L$) not matching a random nonself string a, is given by:

$$p_{\varepsilon^-} = (1 - P(Match(d,a) \mid V(d)))^{\eta_l}$$

$P(Match(d,a) \mid V(d))$ can be computed by a variant of the same algorithm described in section 3.2.4 that computes whether a given nonself string is a hole. Call this algorithm θ (i.e. $theta(d,a,S_r)$ computes $P(Match(d,a) \mid V(d))$), then

$$p_\varepsilon^- = (1 - \theta(d,a,S_R))^\eta$$

The computation of θ is similar to the computation performed for holes, but more expensive: instead of terminating the search of the string space upon finding a valid detector, the searching must continue until all valid detectors have been found. In the worst case this will necessitate $2^{\ell-r}$ search steps, which can be intractable. Instead an approximation is used. Each valid r-template a_r could match $2^{\ell-r}$ strings. Because the $\ell-r+1$ templates in a string overlap each other, this overlap must be taken into account when computing the number of possible matches. The first valid template encountered gives $2^{\ell-r}$ matches, but after that, the matches from a valid template must be reduced according to the overlap. Assume that the i-th template is valid, and that the previous valid template was the j-th template, $i > j$, then the i-th template will give $2^{\ell-r}(1-2^{-\phi_a(i,j)})$ matches, where ϕ is the overlap function and is defined as

$$\phi_a(i,j) = \begin{cases} r & \text{if } i-j > r \\ i-j & \text{otherwise} \end{cases}$$

The total number of matches for a string a is given by the sum of the number of matches for all the $\ell-r+1$ overlapping templates matching a:

$$\text{total matches} = 2^{\ell-r} \sum_{i=1}^{\ell-r+1} \mathit{ValidMatches}_a(i)$$

where

$$\mathit{ValidMatches}_a(i) = \begin{cases} 0 & \text{if the } i\text{-th template is invalid} \\ 1 & \text{if the } i\text{-th template is the first valid template} \\ 1-2^{-\phi_a(i,j)} & \text{if } i\text{-th template is valid and the} \\ & j\text{-th template was the previous valid template} \end{cases}$$

The probability of a match between a valid detector d and a nonself string a is:

$$P(\mathit{Match}(d,a) \mid V(d)) = \theta(d,a,S_R) = \frac{\text{total matches}}{2^\ell} = 2^{-r} \sum_{i=1}^{\ell-r+1} \mathit{ValidMatches}_a(i)$$

If no templates are valid, then $\theta(d,a,S_R) = 0$ and a is a hole. If all templates are valid, then

$$\theta(d,a,S_R) = 2^{-r}(1+(\ell-r)(1-2^{-1})) = 2^{-r}(1+\frac{\ell-r}{2}) = p_{contg}$$

which agrees with the approximation for the matching probability of two random strings under the contiguous bits rule given in equation 3.1.

The expected number of retries can also be more accurately calculated using θ. Without assuming independence, equation 3.2 becomes:

$$\begin{aligned} P(V(d)) &= P(F_1F_2\ldots F_{|S|}) \\ &= P(F_1)P(F_2 \mid F_1)P(F_3 \mid F_1F_2)\ldots P(F_{|S|} \mid F_1F_2\ldots F_{|S|-1}) \\ &= P(F_1)(1-P(F_2^C \mid F_1))(1-P(F_3^C \mid F_1F_2))\ldots \\ &\quad (1-P(F_{|S|}^C \mid F_1F_2\ldots F_{|S|-1})) \end{aligned}$$

where F_i^C is the complement of the event F_i, that is, F_i^C is the event that $\mathit{Match}_h(d,s_i)$. Now the term

$$\begin{aligned} P(F_i^C \mid F_1F_2\ldots F_{i-1}) &= P(\mathit{Match}(d,s_i) \mid \neg\mathit{Match}(d,s_1) \text{ and } \neg\mathit{Match}(d,s_2) \\ &\quad \text{and } \ldots \neg\mathit{Match}(d,s_{i-1})) \end{aligned}$$

is computed by the algorithm $\theta(d, s_i, \{s_1, s_2, \ldots, s_{i-1}\})$. Using a shorthand notation $\theta(d, s_i, \{s_1, s_2, \ldots, s_{i-1}\}) = \theta_i$, the expected number of retries is:

$$E(\rho) = \frac{1}{P(\neg Match(d, s_1)))(1-\theta_2)(1-\theta_3)\ldots(1-\theta_{|S_R|})} \tag{3.7}$$

If the self set is available and is not too large, and a reasonable sample of nonself can be obtained, then the equations and algorithms given here can be used to compute more accurate predictions for real systems when the contiguous bits rule is used. Chapter 4 demonstrates how these theoretical calculations can more accurately predict performance in a real environment, that of network intrusion detection.

3.2.6 Multiple Representations

Holes can exist for any symmetrical match rule with constant matching probability, and the existence of holes places a lower bound on the false negative probability. However, different matching rules generate different holes for the same self set, so it was suggested in [D'haeseleer, *et al.*, 1996] that using different matching rules for different detectors could reduce the overall number of holes. Alternatively (and equivalently), one match rule could be used and the representation changed, because the universe of strings U_R is mapped from the set of events U by a representation, $\Lambda : U \to U_R$. Multiple representations from U to U_R could be used, but in general it is easier for implementation to use a single base representation Λ_B, to map U to a set of base strings U_B, and then to apply every new (or secondary) representation Λ_i to U_B, $\Lambda_i : U_B \to U_i$.

One way of generating a set of different secondary representations is to use *pure permutation*: the bits of the representation are permuted according to a randomly generated permutation mask. For example, given the strings $s_1 = 01101011, s_2 = 00010011 \in U_B$, and a secondary representation Λ_1, defined by a random permutation mask 1-6-2-5-8-3-7-4, then these strings would become $\Lambda_1(s_1) = 00111110$, and $\Lambda_1(s_2) = 00001011$. Using the contiguous bits rule with $r = 3$, under the base mapping, $h(s_1, s_2) = match$, because the last 3 positions are the same, but under secondary representation, $h(\Lambda_1(s_1), \Lambda_1(s_2)) = nomatch$. To implement pure permutation requires storing the random permutation mask, which will require ℓ integers; each integer must be less than ℓ, so $\ell \log_2 \ell$ bits will be required.

However, this pure permutation scheme will not work for the Hamming match rule, because only permuting the bits will not change the Hamming distance between two strings. Furthermore, pure permutation cannot eliminate holes for the contiguous match rule, because the contiguous match rule operates on a metric in Hamming space. A self set of ℓ strings will always suffice to induce holes for the contiguous bits rule with any set of permutation masks. This is shown as follows. A nonself string $a \in N_B$, $a = a_1a_2\ldots a_\ell$ is a hole under the contiguous bits rule (with $r < \ell$, obviously) for any permutation if the base representation of the self set contains every r-template that matches a, which is the case if the self set contains ℓ strings $S_B = \{s_1, s_2, \ldots, s_\ell\}$ such that

$$\begin{aligned}
s_1 &= \overline{a_1}a_2\ldots a_\ell \\
s_2 &= a_1\overline{a_2}a_3\ldots a_\ell \\
&\vdots \\
s_\ell &= a_1a_2\ldots\overline{a_\ell}
\end{aligned}$$

If the bits of a are reordered using a random permutation, then the bits of the strings in S_B will also be reordered, but the set of strings in S_B will still contain every r-template that matches a. Hence a is a hole. However, note that although holes still exist, a larger self is needed to induce them (ℓ strings as compared to

2 strings), so permutation masks lower the bound on false negative probability, even if they do not eliminate it.

Because both the contiguous bits rule and the Hamming rule rely on metrics in Hamming space, a set of representations is needed that moves strings arbitrarily far apart in Hamming space. This can be achieved if each representation is a different, perfect hash function, Λ_{hash}, that *uniquely* hashes strings, that is, for any $s_1, s_2 \in U_B$, $\Lambda_{hash}(s_1) \neq \Lambda_{hash}(s_2)$. Perfect hashing eliminates holes for any $r > 0$, because it is always possible to find a perfect hash function, Λ_{hash}, such that for $a \in N_{hash}$, there exists $d \mid d \notin S_{hash}$ and $Match_h(a, d)$. In general, perfect hashing is too expensive, because the hash function requires a hash table that has an entry for each element in the universe. There are two possible solutions: imperfect hashing and substring hashing.

Imperfect hashing relies on a hash function $\Lambda_{imperfect}$ that is cheaper to store (or compute), but that does not guarantee uniqueness, that is, for any $s_1, s_2 \in U_B$, it is possible that $\Lambda_{imperfect}(s_1) = \Lambda_{imperfect}(s_2)$. The consequence of this is that nonself strings that are hashed to self strings will be holes. In this work the the linear congruential operator [Lehmer, 1949] was used to implement imperfect hashing, because it is efficiently implemented in the form of a random number generator in the C programming language. The linear congruential operator is given by

$$Y_{lcong} = (\alpha Y + \beta) \bmod |U_B| \tag{3.8}$$

where Y is the numeric value of a string $s \in U_B$ and Y_{lcong} is the numeric value of $\Lambda_{lcong}(s)$. The parameters α and β are integers $\alpha, \beta < |U_B|$, and are randomly chosen for each different representation. To implement this representation requires storing the two parameters, each of which are integers less than $|U_B| = 2^\ell$, so at most 2ℓ bits will be required.

With substring hashing, a string is divided into substrings of equal length ℓ_{sub}, and a perfect hash function is applied separately to each substring. The storage requirements are dependent on the length of the substrings. A hash table of $2^{\ell_{sub}}$ entries will be required, where each entry is a binary string of length ℓ_{sub}, so $\ell_{sub}2^{\ell_{sub}}$ bits of information will be required per hash function. If a different hash function is applied to each substring, then the total number of bits will be $\ell 2^{\ell_{sub}}$. If substring hashing is used with the contiguous bits rule, then the substrings can also be permuted with a random permutation mask. This would require an additional $(\ell/\ell_{sub})\log_2(\ell/\ell_{sub})$ bits. Substring hashing is the most expensive representation of those considered here, but it has the property that it guarantees that any random substring hash function will not increase the number of holes over the base representation. The linear congruential operator does not have this property; as we shall see in chapter 4, section 4.3.3, it can increase the number of holes and so increase the number of false negatives.

All of the representations require storage. If every detector used a different representation, this would more than double the space required for the detectors. To avoid this, the representation is applied to each local detection system, that is, each representation is applied to a set of detectors at a single location. Then the computation of the secondary representation has to be performed only once per string for each detector set. Furthermore, for substring hashing the same hash function is used for all substrings within a detector set, so the storage expense is reduced to $\ell_{sub}2^{\ell_{sub}}$ per detector set.

Having a different representation for each local detection system is equivalent to changing the "shape" of the detectors, while keeping the "shape" of the self set constant. If the shape of a detector d is defined by its cover set, C_d, then different representations change C_d. This is illustrated in figure 3.7.

If multiple representations are used, the refined analysis for the contiguous bits rule would have to be performed for every new random representation. Applying the refined analysis to only the base representation would not give accurate predictions. However, the less the representations change the distribution of the self set from the base representation, the more accurate the analysis for retries. Alternatively, if every representation randomly redistributed the self set, then the simple analysis for retries would apply. Furthermore, if the

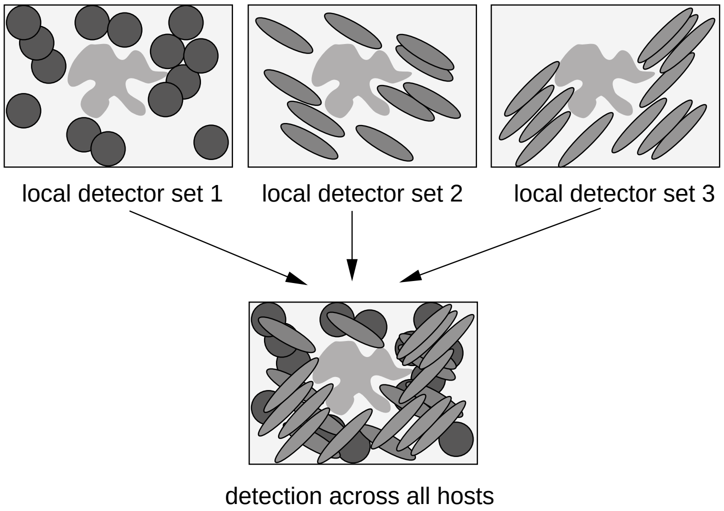

Figure 3.7: Representation changes are equivalent to "shape" changes for detectors. The problem of holes can be ameliorated by using different a representation for each local detection system. Here the shape of the detector represents the cover set generated by a match rule, under some representation. There are different holes for different representations. or equivalently different shaped detectors can cover different parts of the nonself space, for a global reduction in holes.

multiple representations not only randomly distributed the self set, but also randomly distributed the nonself in a local test set, then the simple analysis would apply to the prediction of false negative rates.

It is important to note that we do not necessarily *want* to randomly distribute the self set. The closer the self set is to a random distribution, the greater the number of retries needed to generate valid detectors. In practise, we would like a set of representations such that every representation clumps the self set, while distributing the nonself strings in a local test set randomly.

There is another reason why we may not want to spread the self set out to a random distribution. If a compactly represented self set induces holes, those holes could be close to the self strings in the representation space, because holes are a consequence of the interaction between self strings [D'haeseleer, *et al.*, 1996]. Now, if the training set U_{trn} is *incomplete*, that is, it does not contain all self strings, $\exists s \in S_R \mid s \notin U_{trn}$, and the self set is compactly represented, then it could be that the self strings not in U_{trn} are close to U_{trn} and therefore more likely to be holes. In other words, the holes induced by a compact U_{trn} could increase the tolerance of the system to an incomplete representation of self.

Multiple representations were introduced with the goal of eliminating holes, because what may be holes under one representation may not be holes under another. In effect, multiple representations confer an important kind of diversity. Another form of diversity is conferred by the fact that each local detection system has a set of independently generated random detectors, and so even if one local system does not detect a particular nonself string, it is still possible that another system will.

Because lymphocytes are generalized detectors, it can be assumed that the IS faces a similar problem with holes. It is possible that MHC is a mechanism that implements multiple representations and so ameliorates the problem of holes. This is discussed more fully in section 6.3.1.

3.2.7 Incomplete Training Sets

In the preceding analysis, it was assumed that the training set contained all self strings, $S_R \subseteq U_{trn}$, and hence there could be no false positives. This is not true in the network ID application, because the set of self events changes over time, and the time available in which to collect the self events is too short to collect every event. This is also true in the IS: not all self proteins are expressed in the thymus. In this section the effects of incomplete training sets are analysed.

The first step is to expand the analysis to include the notion of time. It is assumed that all self strings[11] in the training set and in the local test sets are generated by a discrete random process X, which is defined by a set of random variables $\{X_1, X_2, \ldots\}$, where each random variable $X_i, i = 1, 2, \ldots$ is a function mapping events from a sample space to a state space [Grimmet & Stirzaker, 1992]. In this case, the sample space is S_R and the state space is also S_R, so $X_i : S_R \rightarrow S_R$.

It is assumed that X is strongly stationary, meaning that the distribution for all random variables is the same at all times, $P(X_i = s) = P(X_j = s) = p(s)$, for all $i, j = 1, 2, \ldots$. Hence, X is defined by a single distribution. The training self set S_{trn} is collected over a time period $t = 1$ to $t = T$, called the training phase, and every unique string seen during this period becomes an element of S_{trn}, that is, $S_{trn} = \bigcup_{t=1}^{T} X_t$. Note that depending on the distribution of X, a string may be repeated multiple times during the training period. Thus the output of X during the training period defines a sample distribution, $\tilde{p}(s)$.

If this sample distribution approximates the actual distribution of X_t (i.e. $\tilde{p} \approx p$), then the approximate number of false positive errors expected from an incomplete training set can be calculated. The results are derived for a single self test set, $S_l = S_{test}$ at a single location; extension to the global system is straightforward, assuming every detection system is independent. It is assumed that the test set S_{test} is collected after the training set, that is, $S_{test} = \bigcup_{t=T}^{\infty} X_t$. Denote the size of the training set by $m(T) = |S_{trn}|$, and the number of self strings in the test set by $m(\infty) = |S_{test}|$, then $m(T) + m(\infty) = |S_R|$. If the strings in the self set are $S_R = \{s_1, s_2, \ldots, s_{m(\infty)}\}$, then the training set is $S_{trn} = \{s_1, s_2, \ldots, s_{m(T)}\}$ and the test set is $S_{test} = \{s_{m(T)+1}, s_{m(T)+2}, \ldots, s_{m(\infty)}\}$

In the worst case, every self string that is in the test set but not in the training set will be a false positive[12], that is $s_t \in S_{test}$ and $s_t \notin S_{trn} \Rightarrow s_t = \varepsilon^+$, for $t = m(T)+1, m(T)+2, \ldots, m(\infty)$. Given a sample distribution $\tilde{p}$ for X, the probability of seeing a string s_t for the first time after T time steps, $t > m(T)$, is a geometric random variable with parameter $\tilde{p}(s_t)$, so the probability that s_t is a false positive is $P(s_t = \varepsilon^+) = \tilde{p}(s_t)(1 - \tilde{p}(s_t))^T$. The probability of *any* false positive occurring after T time steps is then (this is approximate because $\tilde{p}$ is a sample distribution):

$$P(\varepsilon^+) \approx P\left(\bigcup_{t=m(T)+1}^{m(\infty)} s_t = \varepsilon^+\right)$$

Assuming that the self strings occur independently,

$$P(\varepsilon^+) \approx \sum_{t=m(T)+1}^{m(\infty)} P(s_t = \varepsilon^+) = \sum_{t=m(T)+1}^{m(\infty)} \tilde{p}(s_t)(1 - \tilde{p}(s_t))^T$$

Using an approximation this becomes

$$P(\varepsilon^+) \approx \sum_{t=m(T)+1}^{m(\infty)} \left(\tilde{p}(s_t) - (T-1)\tilde{p}(s_t)^2 + \frac{(T-1)(T-2)}{2}\tilde{p}(s_t)^3\right)$$

[11] In this analysis, nonself strings that may occur in the training and test sets are ignored, because they will, by definition, have no effect on the number of false positives. However, this is an important issue which is addressed later.

[12] Because of generalization, it may be possible that the system will not detect every self string not in the training set.

For T large, $T - 2 \approx T - 1 \approx T$, so

$$P(\varepsilon^+) \approx \sum_{t=m(T)+1}^{m(\infty)} \left(\tilde{p}(s_t) - T\tilde{p}(s_t)^2 + \frac{T^2}{2}\tilde{p}(s_t)^3 \right) \tag{3.9}$$

If the sample distribution approximates the real distribution, and X is actually a strongly stationary process, then equation 3.9 can be used to predict the probability of a false positive, given the number of time steps in the training period and the size of the training set.

3.3 Summary

This chapter showed that for any subset of the universe, there always exists an m-generalization, where m is the maximum Kolmogorov complexity of any pattern in the universe. Furthermore, it was shown that a distributed detection system which makes false positive errors is not scalable in general, whereas an increase in locations can reduce false negative errors, and for a scalable distributed detection system, robustness increases with an increase in the number of local detection systems that are m-generalizations of the self set. These results hold under the following assumptions: the universe is closed; self and nonself are disjoint and partition the universe; every location has sufficient memory capacity to encode the most complex pattern in the universe; if a nonself pattern occurs in a local test set, it occurs in at least one other local test set; the training set is the self set; each local detection system detects at least some of the nonself patterns in its test set; and the number of elements in any local test set is always greater than the number of locations.

Generalized detection is implemented by representing detectors as strings and using string matching. Each local detection system has a set of detectors. Detectors are generated using the negative selection algorithm, which guarantees that a detector is a generalization of the self set, provided the self set is a subset of the training set. In the network ID application, the training sets do not contain all of the self strings; such incomplete training sets will result in false positives. It is possible to predict the false positive rates by assuming that self strings are generated by a strongly stationary random process.

There is a trade-off between the detection rate and the computational requirements for generating detectors. This trade-off is tuned by adjusting the number of detectors and the generality of the match rule. However, the trade-off is limited by holes, which are undetectable nonself strings; as the generality of the detectors increases, the number of holes also increases. Using different representations can reduce the number of holes. Hence, the secondary representation function for each local detection system has randomly determined parameters so that the representations differ between locations. The best representation is one which moves the nonself in a test set furthest from the self to minimize holes, but that keeps the self strings the most clumped together, to minimize the number of retries needed to generate detectors.

Chapter 4

An Application of the Model: Network Security

This chapter describes an application of the abstract model to network security. An Intrusion Detection (ID) system is described that is based upon the Network Security Monitor (NSM) (section 2.3.4), which monitors TCP/IP traffic on a broadcast LAN.

The environment in which NSM was used consisted of a collection of *internal* computers on a LAN, and a collection of *external* computers outside the LAN. Communication between computers (both internal and external) happened via TCP/IP, and was defined by datapath triples (source computer, destination computer, service). A similar environment is used here, with several simplifications:

1. The ID system only monitors network traffic for the presence of unusual datapaths. It does not consider the frequencies of traffic flow over datapaths.

2. The ID system monitors only TCP SYN packets. The start of a connection is indicated by an exchange of TCP SYN packets (see section 2.3.1), so these SYN packets define a pair of datapaths when they first come into existence. All packets following on the same connection use one of the pair of datapaths defined by the SYN packets, so monitoring these non-SYN packets would be superfluous because the ID system only monitors datapaths and not the traffic flow over datapaths (point 1 above).

3. The services are grouped into a few classes. Some assigned services are distinguished (this is explained later), whilst a single port number is used to represent all other unassigned privileged services, and a single port number is used to represent other unassigned non-privileged services.

4. Some services are filtered out from the traffic because it is expected that new datapaths involving these services will always occur; these new datapaths are generally acceptable and should not be regarded as anomalous. For example, all connections to WWW servers are filtered, whether those connections are from external computers to an internal WWW server, or from internal computers to external WWW servers

The first two simplifications reduce the volume of traffic monitored and the size of the self set, hence reducing computational costs. The last two simplifications reduce the probability of acceptable new datapaths occurring in the test set and not the training set, which is desirable because as the probability of new acceptable variations in the test set increase, so the false positive rate will increase. In general, the simpler the data being monitored the better; we should use the simplest possible system that still works. This principle guided our previous research in the domain of host-based ID (see section 2.2).

Section 2.3.5 mentioned several extensions to NSM, including robustness, scalability, flexibility, and adaptability. It was claimed that these properties could be achieved with a distributed ID system, one in which each computer in the LAN has a lightweight detection system. Chapter 3 showed how an architecture based on the immune system can achieve robust and scalable detection, and so the application to network ID should preserve those properties, as indeed it does: it will be shown that the ID system is robust in that every computer contributes towards detection, and scalable, in that, for any given LAN, increasing the number of computers that run detection systems will decrease false negative errors. It will be shown that seven real attack incidents can be detected with local detector sets consisting of no more than 100 string of length 49 bits each, which means that each detector set is lightweight. The ID system is also flexible in that adjusting the number of detectors adjusts the resource usage to error rate trade-off. However, in section 2.3.3, a stricter definition of flexible was used, one that required an ID system to adjust *automatically* its resource usage according to available resources. The ID system was not distributed across a network, rather a network of computers was simulated on a single computer, so this issue was not investigated. However, the abstract model does not preclude this stricter form of flexibility, and chapter 6 describes some ideas for implementing automatic flexibility. Finally, adaptability will be described in the next chapter (5) where it will be shown how dynamic detectors and distributed tolerization are used to respond to changing self sets, and how dynamic detectors, together with memory, can provide the advantages of signature-based detection.

4.1 Architecture

The abstract model can be applied to network ID by mapping the datapath triples to patterns. Then it is assumed that the set of all acceptable or legitimate connections between computers both internally and externally is the self set, and everything else is nonself. The consequences of this assumption are discussed in section 7.3. The patterns or datapaths must have some base representation, for which binary strings of fixed length $\ell = 49$ bits are used. This base representation is described in section 4.1.1.

The internal computers are mapped to n_L locations, and each location can run a local detection system. A local detection system, $\mathcal{D}_l$, at location $l \in L$, consists of a set of detectors D_l, together with a function Λ_l for implementing the secondary representation, and a match rule h_l, that is, $\mathcal{D}_l = (D_l, \Lambda_l, h_l)$. All detector sets are the same size, that is, $|D_l| = |D_j| = n_d, \forall l, j = 1, 2, \ldots, n_L$. Furthermore, all detector sets use the same match rule, $h_l = h_j = h, \forall l, j = 1, 2, \ldots, n_L$. Each detector is a binary string of fixed length $\ell = 49$, drawn from U_R and tolerized using the negative selection algorithm. The detector generation phase is performed off-line, with all detectors being generated in a single location against a single training set. These detectors are then distributed across all locations. When the distributed system is monitoring, all local detection systems see the same traffic because the LAN is broadcast; effectively, all locations have the same test set, $U_l = U_j, \forall l, j = 1, 2, \ldots, n_L$.

This ID system is distributed in the sense that all internal computers in the LAN can run components of the detection system. It is assumed that a uniform policy applies to all computers in the LAN, so the system is distributed within a uniform policy domain. It is not distributed across different policy domains, and so does not address the problems inherent in such distribution.

4.1.1 Base Representation

A SYN packet representing a datapath triple is mapped to a 49-bit binary string (see figure 4.1). The value of 49 bits was chosen as the minimum number needed to represent the relevant information. The composition of the string is as follows. Because at least one computer in the datapath must be on the LAN, the first 8 bits represent an internal computer, and are taken from the least significant byte of its IP address[1]. The following 32 bits represent the other computer involved in the communication, which will require 32 bits for the IP address if that computer is external. If the other computer is internal, then only 8 bits are needed, but 32 bits (the full IP address) are still used to maintain a fixed length representation. If an external computer is involved, it is always represented in the second 32 bits, so an extra bit is used to indicate whether or not the first computer is the server; if the first is the server, the bit is set to one, otherwise it is set to zero.

The final 8 bits represent the type of service. The service type is mapped from its category to a number from 0 to 255. All non-assigned privileged ports are represented by a single number, and all non-assigned non-privileged ports are represented by a different, single number. In the following list, ports are numbered in order, beginning with commonly assigned, privileged ports, followed by commonly assigned non-privileged ports, and finally with a single number for all other privileged ports and a single number for all other non-privileged ports.

- Commonly assigned privileged ports (numbered from 0 to 66): 1, 7, 9, 11, 13, 19, 20, 21, 22, 23, 25, 37, 38, 42, 43, 53, 68, 70, 79, 80, 87, 94, 95, 109, 110, 111, 113, 119, 123, 130, 131, 132, 137, 138, 139, 143, 156, 161, 162, 177, 178, 194, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 210, 213, 220, 372, 387, 396, 411, 443, 512, 513, 514, 515, 523, 540, 566
- All other privileged ports: (numbered 67) 1023
- All other non-privileged ports: (numbered 68) 1024
- Commonly Assigned Non-privileged ports (numbered 69): 6000–6063

There are only 69 numbered services, which means only 7 bits are required for their representation, not 8 bits as shown in figure 4.1. The extra bit is included to accommodate numbering for more services.

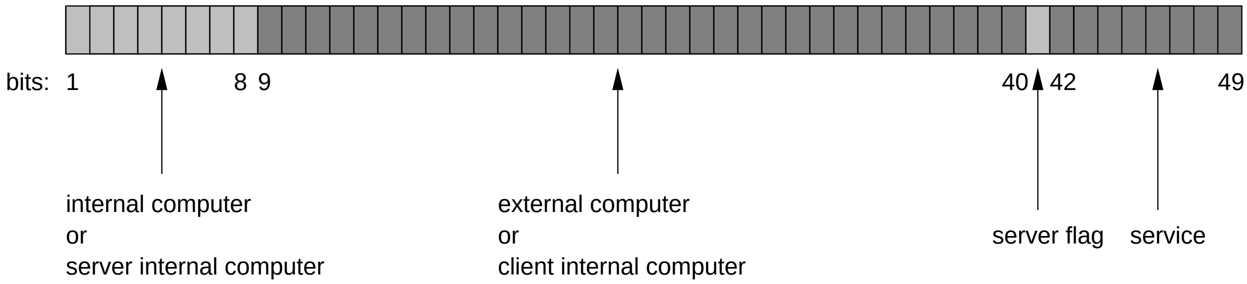

Figure 4.1: Base representation of a TCP SYN packet.

There are several categories of datapaths that are filtered out:

WWW and ftp servers: Traffic to WWW and ftp servers is varied, and it is acceptable for multiple new computers to connect to the server every day. Definitions of self in terms of datapath triples will be useless for distinguishing normal from anomalous behaviour. Other methods will be needed to monitor abuses of WWW and ftp servers, possibly methods that study the contents of packets in addition to the

[1] It is assumed here that all computers on the LAN have the same class C network address. This representation is thus limited in that there can be at most 256 computers on the LAN, which is true for the network used to collect data for all experiments in this dissertation.

source and destination. However, within the set of internal computers, only one or a few will be set up to be WWW or ftp servers, so if any other internal computers suddenly begin functioning as WWW or FTP servers, this is suspect, and hence this kind of traffic *is* monitored.

Dynamic IP addresses: If any service (for example, telnet) on an internal computer is frequently connected to by computers that have dynamic IP addresses (such as those allocated by internet service providers to dial-up clients), then that service could experience a new connections, because the same client computer could be allocated a different IP address on each dial-in. This will create a new datapath on every connection to the service, even though the same two computers are always communicating. It is possible that with the correct generalizations, dynamic IP addresses would not be a problem, but here this traffic is filtered out.

Unusual internal computers: In the data collected for these experiments, there was one internal computer that was excluded because a system administrator used it as a base for periodically and randomly portscanning internal computers. Such behaviour is legitimate, but new legitimate connections can occur at any time, so it is not amenable to characterization by datapaths.

4.1.2 Secondary Representations

Three types of secondary representation functions were described in section 3.2.6: pure permutation, substring hashing, and imperfect hashing. Each of these defines a parametric family of representations. The same family, Λ_{family} is used for all local detection systems, but the parameter sets Φ are randomly determined, so effectively each local detection system has a different secondary representation, determined by $\Lambda_{family}(\Phi)$. The parameters for each family are:

Pure permutation: $\Phi = \{x\}$, where $x = x_1 x_2 \ldots x_{49}$ is a permutation mask, that is, each x_i is an integer, $1 \leq x_i \leq 49$, indicating the position to which the current bit value must be mapped. For example, if $x_3 = 10$, then the bit in position 3 must be mapped to position 10. Note that x is randomly generated such that $x_i \neq x_j, \forall i, j = 1, 2, \ldots, 49$, that is, the x_i are generated without replacement.

Imperfect hashing: $\Phi = \{\alpha, \beta\}$, where α and β are the parameters for the linear congruential operator (see equation 3.8). In this implementation, the 32-bit C-language random number generator was used for efficiency; it is defined by two C functions: *lcong48* and *lrand48*. The first function, *lcong48*, initializes the random seed, using the parameters α (a 48 bit number), β (a 16-bit number) and Y (a 48-bit number). *lrand48* has no input; it returns a 32-bit number. These two functions map the 48-bit number Y to a 32-bit number, which is the output of *lrand48*. So to map a 49-bit string s to a secondary representation s_{lcong}, the following sequence of calls is made: first, $lcong(Y, \alpha, \beta)$, where Y is the decimal value for the first 48 bits of s, then $lrand48$ for the first 32 bits of s_{lcong}, and $lrand48$ again, using the first 16 bits returned for the last 16 bits of s_{lcong}[2]. For efficiency, the final 49-th bit is mapped across with no change.

Substring hashing: $\Phi = \{x, ht\}$, where x is a permutation mask and ht is a hash table. The substrings for the perfect hash function are 8 bits in length, so $ht = ht_1, ht_2, \ldots, ht_{256}$ is an array of 256 unique random values, $ht_i \neq ht_j, \forall i, j = 1, 2, \ldots, 256$, and $x = x_1 x_2 \ldots x_6$ is a permutation, $1 \leq x_i \leq x_6$, $\forall i = 1, 2, \ldots, 6$. Each of the 6 bytes in a 49-bit string s is permuted using x, and then these bytes are hashed to new byte values using ht. The 49-th bit is mapped across with no change. Figure 4.2 illustrates this representation.

[2]Two calls are required to *lrand48* because it only returns a 32 bit number, in spite of its name. In C syntax, $s_{lcong} = lrand48\&(lrand48 >> 32)$.

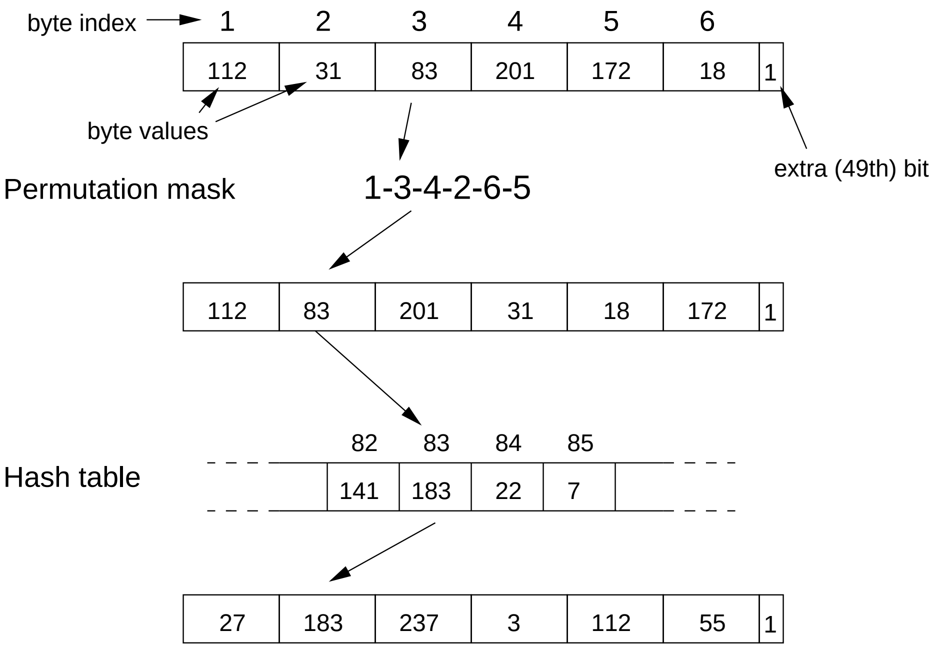

Figure 4.2: Substring hashing. A binary string s of length $\ell = 49$, consists of 6-byte values plus an additional bit. These bytes are permutated via the permutation mask, $x = 1-3-4-2-6-5$, which is randomly generated, and then s is remapped via a randomly generated hash table, ht. The additional bit is not remapped.

4.1.3 Activation Thresholds and Sensitivity Levels

Section 3.2.7 analysed the case where the training set does not include all self strings. Such a circumstance will result in false positives, which, as shown in section 3.1, cannot be tolerated if we want a system that is scalable. If a complete training set cannot be collected, then false positives must be minimised in some other way. Two mechanisms are introduced to reduce false positives. These mechanisms are based on two assumptions, both of which hold for the real network data that are reported in sections 4.2 and 4.3. The assumptions are:

1. An incident will generate multiple nonself strings. This is true for the real incidents described in section 4.2, where each of seven incidents consists of at least 36 nonself strings.

2. When an incident occurs, it will generate nonself strings at a greater rate than the rate of occurrence of new self strings, over some given period of time.

These assumptions imply that nonself strings tend to occur in temporal clumps relative to new self strings. As the number of connections used by an attacker is reduced, so the difficulty of attacking increases, because the more connections there are, the more information an attacker can gain about the target network. Furthermore, the more the attacks are separated in time, the longer it will take to execute an attack, requiring more patience and possibly skill from an attacker. As these assumptions are increasingly violated, so the mechanisms for reducing false positives will be of decreasing effectiveness.

The two mechanisms for reducing false positives are:

Activation thresholds: This mechanism is based on the concept of affinity thresholds in the immune system: multiple receptors on a lymphocyte must bind to epitopes before that lymphocyte is activated (see section 2.1.1). An analogous mechanism is used in which each detector has to match a certain number of times before it is activated and signals an anomaly. Each detector $d \in D_l$ has a match count c_d, which is incremented every time d matches a string. When c_d exceeds the activation threshold, τ, an anomaly is signaled and the counter for d is reset to 0, that is, $c_d = 0$. The activation threshold, $\tau \geq 1$, is a global parameter, that is, it is the same for all locations. This mechanism has a temporal horizon: the match count c_d decays at a rate determined by a decay parameter, γ_{match}, which indicates the probability of the match count dropping by one. Decay is modeled by a negative binomial random variable, with c_d/γ_{match} expected time steps until the match count for d is reduced to 0. The effect of this mechanism is that anomalies will be detected only if they are caused by groups of similar strings that occur sufficiently frequently within a limited time period.

Sensitization: This mechanism is intended to improve detection of DCAs and similar attacks. Activation thresholds are not sufficient, because they require that a single detector match repeatedly, and if the attacks are launched from different sources, a single detector may not match all the attacks. This additional mechanism is intended to "sensitize" the detection system so that a a burst of connections from multiple *different* locations will generate anomalies. This is analogous to the effect of cytokine levels in the IS. When suspicious activity is detected by some cells of the IS, cytokines are released which alert (or effectively "sensitize") the rest of the IS. Each detection system $\mathcal{D}_l$ has a sensitivity level σ_l, which is a non-negative real. Every time the match count for a local detector $d \in D_l$ goes from 0 to 1, the sensitivity level is increased by a factor ω, that is, $c_d(t) = 0$ and $c_d(t+1) = 1 \Rightarrow \sigma_l(t+1) = \sigma_l(t) + \omega$, for $t = 1, 2, \ldots$. This mechanism also has a temporal horizon: the sensitivity σ_l decays over time with a decay parameter, γ_ω, which is the probability that the sensitivity will drop by 1. The sensitivity affects local detectors by reducing the number of matches needed for activation: detectors at location l will only need $\tau - \sigma_l$ matches to be activated. The sensitivity can never reduce the activation threshold below one, that is $0 \leq \sigma_l < \tau, \forall l = 1, 2, \ldots, n_L$. The effect of this mechanism is that anomalies occurring within a limited time period, even if they involve different strings, can still be detected.

Both of these mechanisms are tunable. If security is important and an organization has enough resources to cope with higher false positive rates, then the sensitivity factor ω and the activation threshold τ can be set low. Conversely, if security is not so important, and minimizing false alarms is, these parameters can be set higher.

4.2 Experimental Data Sets

The ID system was tested with a simulation, using data gathered at the University of New Mexico (UNM), from a subnet at the Computer Science department. The subnet consisted of 50 computers on a single segment; these 50 computers were simulated on a single computer. Data were collected using the tcpdump program from the subnet for a period of 50 days, to yield a total of 2.3 million datapaths (TCP SYN packets). After filtering out WWW, ftp and other traffic (as described in section 4.1.1), the 50 days of traffic mapped to 1.5 million strings. These data were used for both self training and test sets, and nonself test sets. All nonself traces were manually extracted from the data, thus separating the self and nonself patterns.

4.2.1 Self Sets, S_{trn} and S_{test}

Of the 1.5 million strings, 1.27 million were used as the training set, S_{trn}, corresponding to the first 43 days of network operation, and the remaining 182000, corresponding to the last 7 days, were used as the self test set, S_{test}. The training set consisted of 3763 unique strings, $|S_{trn}| = 3763$, and the self test set consisted of 869 unique strings, $|S_{test}| = 869$, of which 137 were not found in S_{trn}. The distribution of the self strings (training plus test sets) was analysed to determine if there is an approximation to a strongly stationary process underlying the system. The results are shown in figure 4.3, which is a plot of the sample distribution of the self set, $p(s) = P(X_t = s)$, $s \in U_{trn} \cup U_{test}$. The 200 most frequent strings account for approximately 95% of the strings occurring. Of the remaining 3700 less-frequent strings, their distribution is accurately described by a power-law of the form

$$p(s_i) = ai^{-b} \tag{4.1}$$

where $a = 5.54$, $b = 2.0$ and $200 \leq i \leq 3900$ is the index of the i-th most frequent string. It was stated in section 3.2.7 that a strongly stationary process, $X = \{X_1, X_2, \ldots\}$, is one in which the random variables at each time index have the same distribution, that is, $P(X_i = s) = P(X_j = s) = p(s)$ for all i, j. This power-law can be used to model a strongly stationary process where the probability of occurrence of any string is given by equation 4.1.

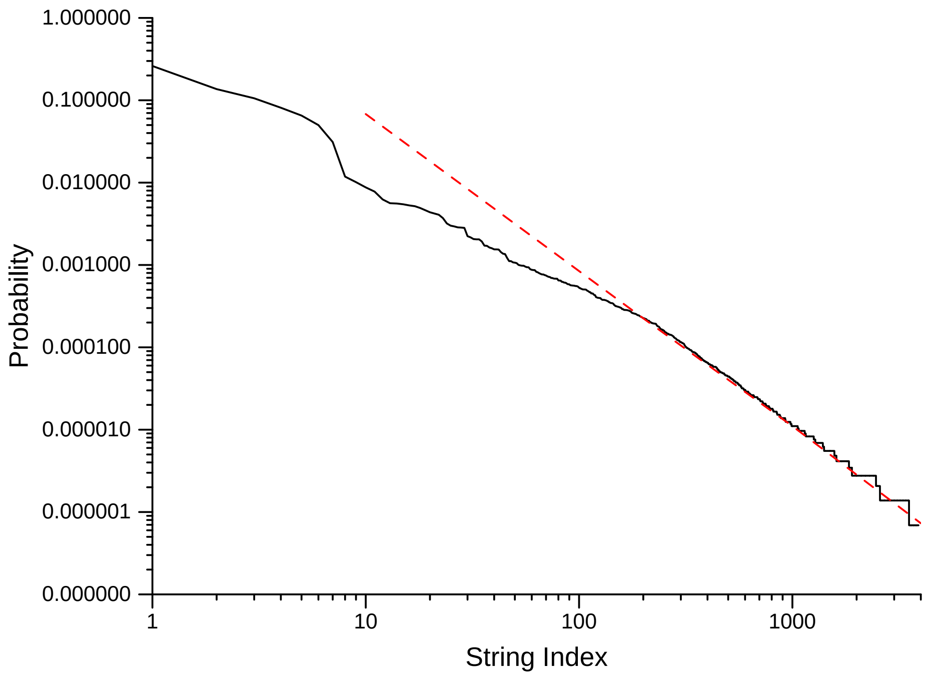

Figure 4.3: Sample distribution of self strings. The unique strings are ranked by frequency of occurrence, and the probability is computed as the frequency normalized to a value between 0 and 1. The strings are plotted in order, from the most frequent at string index 1, to the least frequent at string index 3900. The dotted line indicates the power-law, $p(s_i) = 5.54i^{-2}$. Note that both axes are logarithmic.

4.2.2 Nonself Test Sets, N_{test}

A total of 10 nonself test sets were used. Each test set contained a different intrusive incident. The first seven of the test sets are faithful logs of real incidents, extracted from the 50 days of network traffic, and the last three were synthetically generated. The incidents are:

Address Probing (AP): An internal computer was compromised, and then the intruders used the internal computer to address probe many external domains. About 4500 external computers were probed, all on the telnet port (23).

Portscanning (PS): An external computer, cartan.cas.suffolk.edu, portscanned 33 internal computers, trying to connect to all ports.

Limited probing 1 (LP1): An external computer, xtream.fce.unal.edu.co, attempted to connect to four ports, telnet (23), smtp (25), dns (53) and pop (110), on 35 internal computers.

Limited probing 2 (LP2): An external computer, pc35.esp.ele.tue.nl, portscanned 30 internal computers, trying to connect to 3 ports: telnet (23), dns (53) and imap (143).

Limited probing 3 (LP3): An external computer, sauron.atcomp.cz, made repeated attempts to connect to three ports, telnet (23), dns (53) and imap (143), on a single internal computer.

Single port probing 1 (SP1): An external computer, phear.cwave.com, probed tcpmux (port 1) on 30 internal computers.

Single port probing 2 (SP2): An external computer, dt031n93.midsouth.rr.com, probed the imap port (143) of 30 internal computers.

Synthetic internal (SI): This test set consisted of 1000 random connections between internal computers. The source and destination computers were chosen randomly from all available internal computers, and the service was chosen randomly from a list of common services, namely, ftp (21), telnet (23), smtp (25), and finger (79). This test set can be regarded as modeling a collection of attacks in which the attacker already has access to at least one internal computer, or it can be regarded as a model for the pattern of intrusions caused by a worm propagating across the LAN, via a few vulnerable services.

Synthetic external (SE): This test set consisted of 1000 random connections between external computers and internal computers, on a few common services (ftp, telnet, smtp, finger). One computer in the connection was a randomly selected internal computer, and the other was a randomly selected external computer. This test set can be regarded as a simulation of a Distributed Coordinated Attack (DCA), because it involves many different attacking computers.

Random (RND): This test set consisted of 1000 randomly generated strings of length 49 bits.

Most of the incidents involved probing of telnet (AP, PS, LP1, LP2, LP3), imap (PS, LP2, LP3, SP2) and dns (PS, LP1, LP2, LP3) ports. Within the last six months of gathering this data, vulnerabilities in imap and dns have been publicized; clearly, the attackers were looking for the most recent security flaws. Presumably the telnet port was being probed for weak passwords or default accounts. The probing on port 1 is a method for determining if the target computer is an SGI, because SGIs usually respond in a particular way to requests on port 1.

For the real incidents, the detection system was tested against the trace of the incident, which consisted of all network traffic between (and including) the first and last nonself SYN packets. Thus these real

Test Set	Number strings	Fraction nonself	Fraction unique nonself
AP	8600	0.540	0.340
PS	2966	0.435	0.196
LP1	1174	0.617	0.118
LP2	114	1.000	0.842
LP3	1317	0.102	0.002
SP1	36	1.000	0.833
SP2	285	0.165	0.130
SI	1996	1.000	0.997
SE	1996	1.000	1.000
RND	3763	1.000	1.000

Table 4.1: Features of nonself sets. *Number strings* is the total number in the trace, from first nonself string to last, including all self strings that occurred during that time. *Fraction nonself* is the fraction of the trace that consisted entirely of nonself strings, and *Fraction unique nonself* is the fraction of the trace represented by unique nonself strings, which is the size of the nonself test set for the incident.

incidents reproduced the timing of the attack, as well as including all normal traffic that was interspersed throughout the attack. Table 4.1 summarizes the total number of strings in the trace of each incident, the number of strings that are nonself, and the number of unique nonself strings, which is the size of the nonself test set for that incident. The synthetic test sets were created to test particular aspects of the model. The simple theory relies on a random nonself test set, so a random set was created to test this theory. The SI test set consisted of nonself strings that are closer to self than random strings, and so should be harder to detect than random strings, and hence the predictions of the simple theory should be less accurate. The SI test set was included as an extreme case to determine how proximity to the self set affects the simple theory.

4.3 Experimental Results

Several experiments were conducted using these data sets. For all experiments, a false negative error refers to a single nonself string that was not detected, and a false positive error refers to a single self string that was detected (any string that is detected is classified as *anomalous*). False positives are reported as error rates, where the error rate, $\Delta\varepsilon$, is the number of errors normalized over the length of the trace, or over some time period Δt (usually a day), that is,

$$\Delta\varepsilon^{+} = \frac{1}{\Delta t}\left|\{s \mid s \in \{X_t\} \text{ and } s = \varepsilon^{+}\}\right|, \forall t \in \Delta t$$

Detection rates are reported instead of false negative error rates, where the detection rate, ψ, is the number of successful detections of nonself over the length of the trace, that is, $\psi = 1 - \Delta\varepsilon^{-}$.

These definitions of error rates and detection rates differ from those normally used in ID. There a false negative refers to an intrusive incident that was not detected. By contrast, in this dissertation, a false negative refers to a single connection (string) that was not detected. Under the assumption that an intrusive incident will involve several connections, it suffices to detect only some of the connections to detect the incident. In the simplest case, if $\psi > 0$ for a single incident, the incident will be detected. We want ψ to be as much above zero as possible, because the higher the value of ψ, the clearer the separation between the incident and possible false positives. ψ can be thought of as the *strength of the anomaly signal*.

Parameter	Description	Default Value
n_L	number local detection systems	50
n_d	number detectors per location	100
ℓ	string length	49
r	match length	12
Λ	secondary representation	substring hashing
h	match rule	contiguous bits
τ	activation threshold	1
γ_{match}	match decay	0
ω	sensitivity	1.5
γ_ω	sensitivity decay	10

Table 4.2: The parameters for the basic distributed ID system. Unless stated otherwise, all experiments use these default values. Note that when $\tau = 1$, the settings for γ_{match}, ω, and γ_ω are irrelevant, because τ can never be less than 1

A summary of all the parameters, together with their default values, is given in table 4.2. These parameter settings were chosen according to pilot studies and the experimental results reported later in this chapter. Parameter settings for experiments are always stated in the caption of a figure or table. Usually only a few parameter settings are given; those omitted have the default values. Where multiple runs with different random seeds are performed for an experiment, the usual number is 30, unless otherwise stated. For experiments that involve multiple runs, the mean and a 90% confidence interval in the mean are reported, unless stated otherwise. The confidence interval is indicated by error bars in figures, and by $\pm$ signs in tables.

For most of the experiments in this chapter, the synthetic nonself test sets were used, because the structure of these test sets is known, and they represent extremes of nonself test sets. This makes it easier to understand and interpret results. The various aspects of the model are explored using the synthetic data, and then section 4.3.5 reports results for experiments with the real nonself test sets.

4.3.1 Generating the detector sets

The first experiment determined how many retries are needed to generate detectors, and tested if the theories developed in chapter 3 are good predictors of the empirical results. No secondary representations were used for this experiment. To test the simple theory (equation 3.4), a random self training set was generated, one of the same size as the real or structured training set, that is, 3763 strings. The number of retries per detector was computed for 200 detectors, for two cases: tolerization against the random training set, and tolerization against the structured training set. The results are shown in figure 4.4. The simple theory more accurately predicts the number of retries against the random self than against the structured training set. However, the refined theory (equation 3.7) more accurately predicts the retries for the structured self. The important point about this figure is that the structure in the training set reduces the number of retries required over a random training set; for example, when $r = 13$, the number of retries against the random set is 6200, but against the structured set it is only 4, which is 3 orders of magnitude less. This emphasizes the point that we do not want the self set to be randomly distributed across the string space, because then detector generation is more expensive.

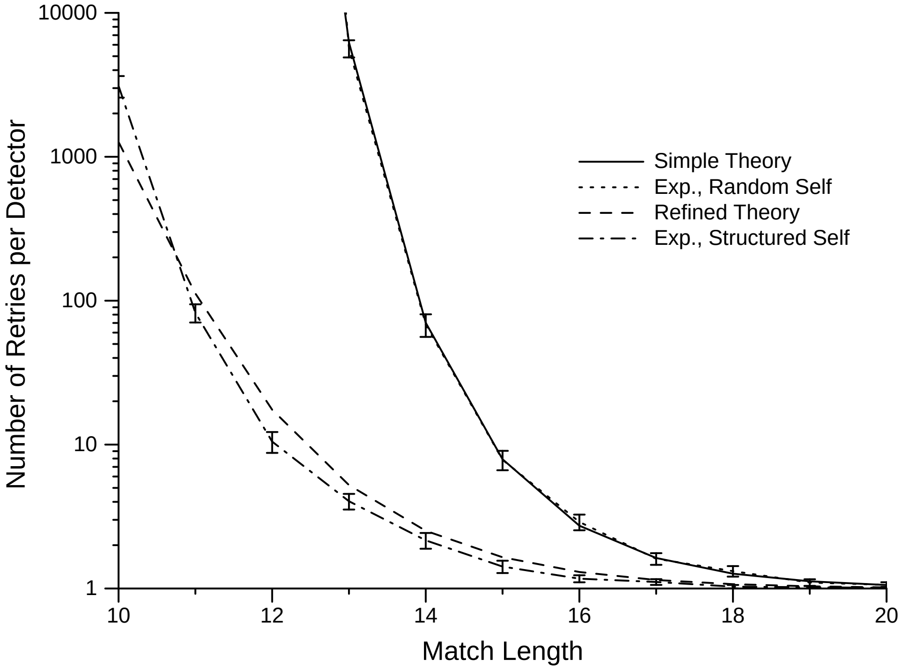

Figure 4.4: Expected number of retries, $E(\rho)$, for tolerization versus match length, r (r varies, Λ = none). The plot shows predictions given by equation 3.4 (*Simple Theory*) and equation 3.7 (*Refined Theory*), and experimental results with a random self training set (*Exp., Random self*), and with the structured self training set (*Exp., Structured self*). Note that the y-axis is logarithmic.

4.3.2 Match Rules and Secondary Representations

One way to evaluate different match rules and different secondary representations is in terms of the trade-off between detection rates and retries[3]. The best combination of match rule and secondary representation will be that which minimizes retries and maximizes detection rates for a fixed number of detectors. The two match rules, contiguous bits and Hamming were tested with each of the secondary representations, imperfect hashing (using the linear congruential operator), and substring hashing. In addition, the contiguous bits rule was tested with pure permutation. The results for SE and RND are shown in figures 4.5 and 4.6, respectively. With these plots, the higher the curve the better, because this implies higher detection rates with fewer retries (the best point is in the top left-hand corner). For these test sets, the secondary representation does not help at all; in the case of linear congruential hashing, performance is worse (the curve is lower), because this imperfect hashing can introduce new errors by mapping several strings to a single string.

These curves can be matched with a slight modification to the simple theory. Rearranging equation 3.6 gives

$$\psi = 1 - E(\rho)^{-\frac{n_d}{|S_{trn}|}\beta} \tag{4.2}$$

where β is a constant. The value of β is given in parentheses after the *Theory* label in the two figures (4.4 for SE, and 5.5 for RND). This modification implies that the performance of the system with most of the match rules and secondary representations can be modeled as if S_{trn} were smaller and/or n_d larger. β can be regarded as an indication of how close the test set is to the training set in match space: the higher the value of β, the more random the test set compared to the training set.

For SI, all the secondary representations improve performance (they have higher curves), except for pure permutation, which is to be expected, because (as shown in section 3.2.6), holes still can exist for pure permutation. Pure permutation does worse than having no permutation at all with the contiguous bits match rule; this is a consequence of the pure permutation spreading out the training set (and hence increasing retries), but not improving detection because holes are not eliminated. Although the linear congruential operator improves performance when used with the Hamming match rule, it is not as good as contiguous bits with either the linear congruential operator, or substring hashing. Once again, the simple modified theory predicts performance, with $\beta = 1$. Thus the modification is unnecessary, which implies that although the training set can be regarded as smaller because of similarities in the set, the effect of the numbers of detectors is also reduced because of the proximity of the test set to the training set. This implies that these two effects cancel each other out. What this indicates is that we can use the simple theory, without modifications, as a lower bound on expected performance.

The conclusion to draw from these experiments is that the contiguous match rule, combined with substring hashing works the best in all the cases tested here. In the remainder of the experiments reported in this dissertation, this combination is used: contiguous match rule with substring hashing (note that these are the default parameter settings in table 4.2).

4.3.3 The Effects of Multiple Secondary Representations

It was shown in the previous subsection that having multiple secondary representations improves performance when the nonself test set is "close" to the training set. In this section the notion of closeness is defined and measured. The contiguous bits rule is based on a metric in Hamming space, so Hamming distance can be used to measure closeness. For each nonself string $a \in N_{test}$, the minimum Hamming distance to the training set is determined (the smallest Hamming distance between a and all $s \in S_{trn}$), and then the average over

[3] If detector generation is rare, and resources are not limited, then it may be better to evaluate match rules only in terms of detection rates.

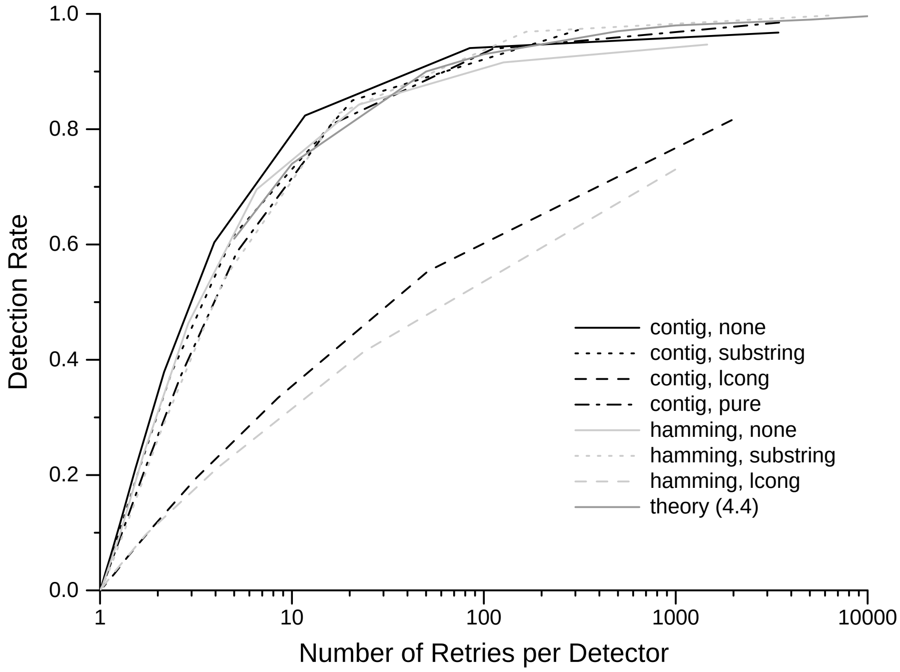

Figure 4.5: Trade-offs for different match rules and secondary representations on SE ($n_d = 10$, r varies, Λ varies, h varies). *contig* refers to the contiguous bits rule, *hamming* refers to the Hamming match rule, *none* means no secondary representation, *pure* refers to pure permutation, *substring* refers to substring hashing, and *lcong* refers to the linear congruential implementation of imperfect hashing. The theory is derived from equation 4.2 with $\beta = 4.4$. Note that the x-axis is logarithmic.

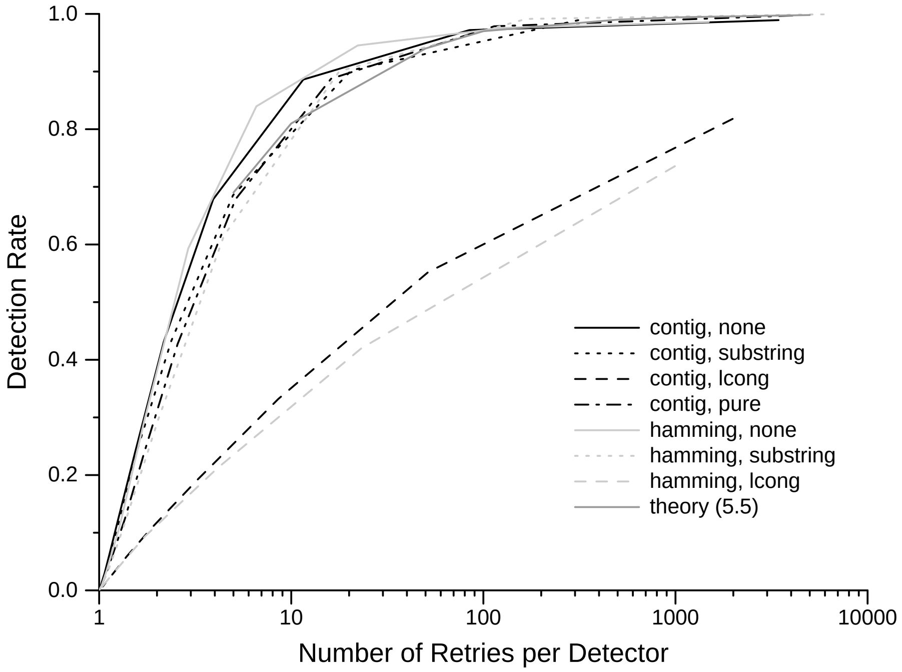

Figure 4.6: Trade-offs for different match rules and secondary representations on RND ($n_d = 10$, r varies, Λ varies, h varies). *contig* refers to the contiguous bits rule, *hamming* refers to the Hamming match rule, *none* means no secondary representation, *pure* refers to pure permutation, *substring* refers to substring hashing, and *lcong* refers to the linear congruential implementation of imperfect hashing. The theory is derived from equation 4.2 with $\beta = 5.5$. Note that the x-axis is logarithmic.

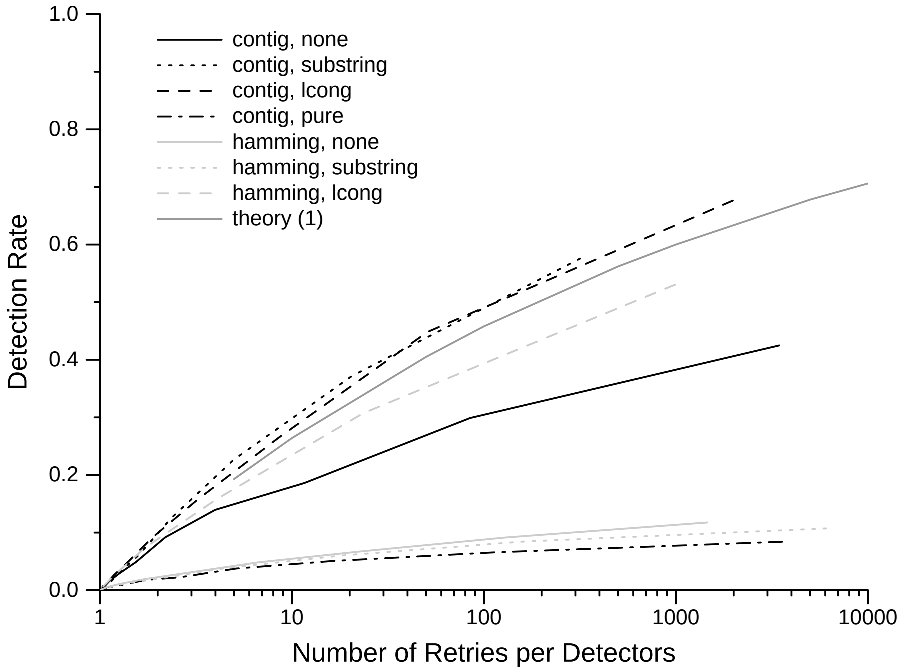

Figure 4.7: Trade-offs for different match rules and secondary representations on SI ($n_d = 10$, r varies, Λ varies, h varies). *contig* refers to the contiguous bits rule, *hamming* refers to the Hamming match rule, *none* means no secondary representation, *pure* refers to pure permutation, *substring* refers to substring hashing, and *lcong* refers to the linear congruential implementation of imperfect hashing. The theory is derived from equation 4.2 with $\beta = 1.0$. Note that the x-axis is logarithmic.

Test Set	nearness	Ratio of Improvement in ψ	
		substring	lcong
SI	0.93	1.60	3.46
SE	0.74	0.99	0.85
RND	0.72	1.00	0.77

Table 4.3: The effects of nearness ($n_d = 10$, Λ varies). Nearness is calculated according to equation 4.3. *substring* refers to substring hashing, and *lcong* refers to the linear congruential operator. The ratio of improvement in ψ is the ratio of ψ when using multiple secondary representations, to ψ when only using the base representation, $Ratio = \psi_{sec}/\psi_{base}$. The base detection rate is 0.24, that is, $\psi_{base} = 0.24$.

all the nonself strings in the test set is computed. The nearness is then the string length minus this average, normalized:

$$nearness = \frac{1}{\ell}\left(\ell - \frac{1}{|N_{test}|}\sum_{a \in N_{test}} \min_{s \in S_{trn}} H(a,s)\right) \tag{4.3}$$

where $H(a,s)$ is the Hamming distance between a and s.

The higher the nearness, the closer the test set is to the training set, so it is expected that the higher the nearness, the more multiple secondary representations will improve performance, and when nearness is low enough, the effects will be negligible. This is indeed the case, as shown in table 4.3. As expected, SI is the nearest, with both SE and RND further away (roughly the same nearness values). The table shows the effect of multiple secondary representations on detection of a given test set. Linear congruential has the most effect on the SI (3.46 times the base representation detection rate) because it spreads the self set out the most; substring hashing has less effect but still improves the detection rate by a factor of 1.6. Although this shows that linear congruential is better for SI, it also makes it more difficult to generate detectors and so the advantage is cancelled, as was previously shown in figure 4.7. For SE and RND, substring hashing makes no difference: the detection rate is the same as for the base representation (the ratio is one). However, for SE and RND, using linear congruential results in a reduction in detection rate (the ratio is 0.85 and 0.77 of the base), because the imperfect hashing scheme introduces more errors by mapping several strings to one string.

Most local detection systems contribute to global detection rates, as is shown in figure 4.8. This figure gives a frequency histogram, indicating how much each local detection system contributes to the global detection rate against SI. Almost all of the local detection systems contribute; few have $\psi = 0$. Furthermore, most locations contribute similar amounts; the best is only 60% better than the median, and is not high ($\psi = 0.17$) compared to the global detection rate for this problem, which is 0.94. These results imply that the system is robust: loss of a few locations will not result in complete failure of detection. These experimental results give support for assumption 6 made in section 3.1.3 (all local detection systems have a non-zero detection rate).

It is possible to predict detection rates when multiple secondary representations are used. In the case where the nonself test set is randomly distributed, the simple theory still holds, as shown by the upper curve in figure 4.9. However, this theory does not apply in the case of SI, but nonetheless, the curve for experimental detection rate on SI can be defined by the same function, modified by a linear transformation on the matching length, r, so that equation 3.1 becomes

$$p_M \approx 2^{-rk-c}\left(\frac{\ell - r}{2} + 1\right) \tag{4.4}$$

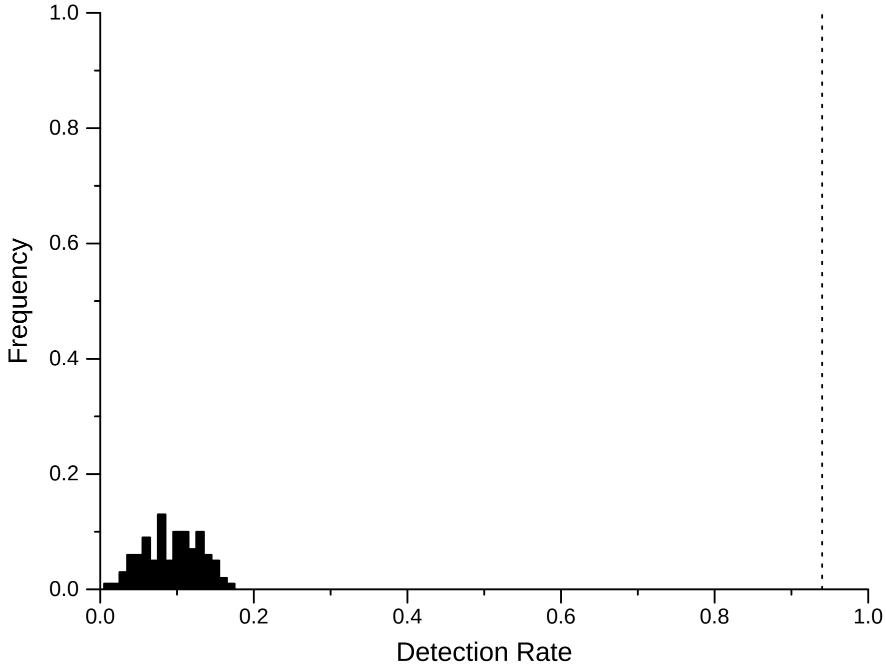

Figure 4.8: The distribution of detection rates on SI ($n_L = 100$). The frequency on the y-axis indicates the fraction of locations with the corresponding detection rate given on the x-axis. The global detection rate was 0.94, and is indicated by the vertical dotted line.

where k and c are constants.

As shown in figure 4.9, this modified match probability with $k = -0.895$ and $c = 3.67$, used in equation 3.5 gives accurate predictions of the detection rate for SI. Equation 3.5 is derived assuming that the nonself strings are randomly generated, and hence the probability of detection of one nonself string is independent of the probability of detection of other nonself strings. The fact that the prediction is still accurate, with a linear transformation to equation 4.4, implies that we can model the probability of detection of the nonself strings by assuming independence, but with a lower probability of matching than that predicted by equation 3.1. This is justifiable because SI consists of randomly generated internal connections, so the resulting nonself strings should be random with respect to each other, but harder to detect because of their proximity to the self set.

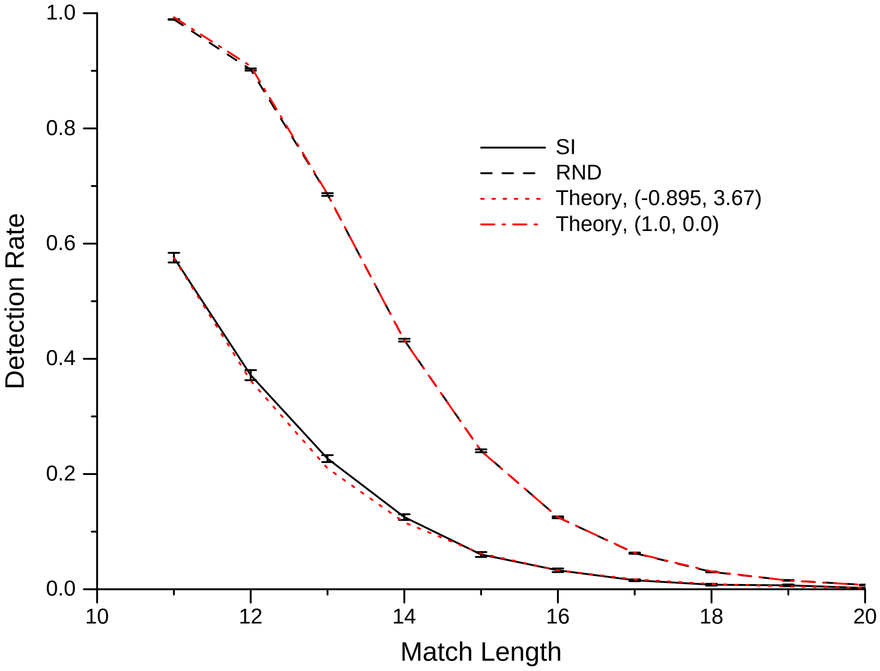

Figure 4.9: Predicting ψ using the modified simple theory ($n_d = 10$, r varies). The theory curves are derived using equation 4.4, and the numbers in parenthesis after the *Theory* labels indicate the parameter settings for k and c, respectively.

This modified theory does not continue to give accurate predictions as the number of detectors is increased, as shown in figure 4.10. This figure shows the results for detecting SI when increasing the size of the local detector sets, n_d. For low n_d, the modified theory is accurate. However, as n_d increases, the theory becomes less accurate, consistently predicting a detection rate that is too high. A possible explanation is that the theory is becoming more inaccurate as n_d increases because as ψ approaches 1, it becomes harder to detect the few remaining nonself strings, so the theory that adding detectors will reduce errors exponentially no longer holds, because most added detectors will not cover those few remaining strings. The assumption of independent detectors no longer holds because the domain is a closed world. It is also possible that those few remaining strings are holes for all secondary representations, so the detection rate is limited. If this is the case, there are still fewer holes than for the base representation: using the algorithm given in section 3.2.4, the hole limit on the detection rate for SI is computed as 0.81, so, for all $n_d > 50$, having multiple secondary representations has overcome this hole limit.

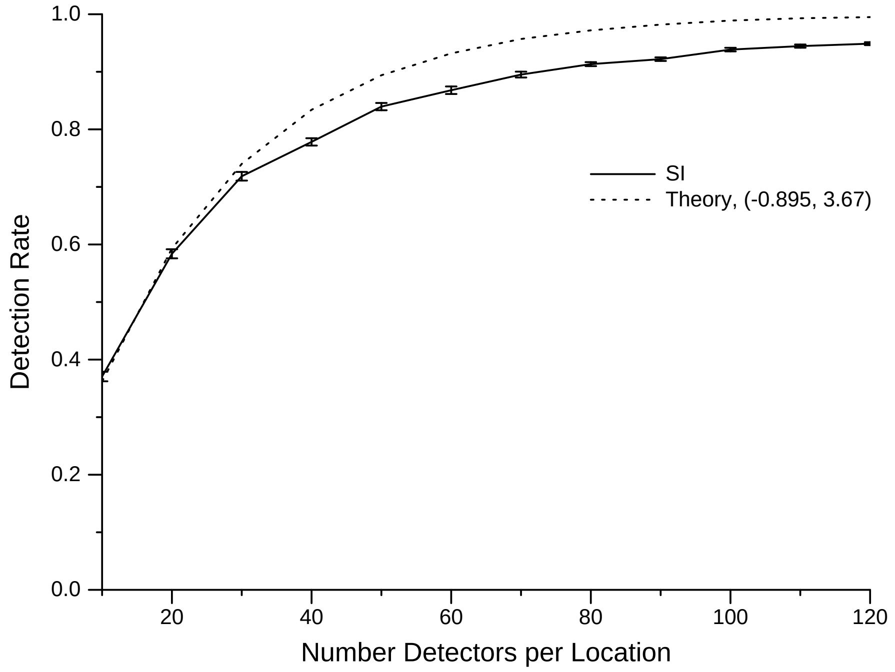

Figure 4.10: Predicting ψ for SI using the modified simple theory (n_d varies). The theory curve is derived using equation 4.4, and the numbers in parenthesis after the *Theory* label indicates the parameter settings for k and c, respectively.

4.3.4 Incomplete Self Sets

Section 3.2 analysed the case where the training set is incomplete, that is, $\exists s \in S \mid s \notin S_{trn}$. Here, that analysis is applied to the experimental data for the purpose of predicting false positive rates. In section 4.2.1 a power-law was fitted to the tail of the sample distribution of self (equation 4.1). This distribution can be substituted into equation 3.9, because for the analysis only the tail of the distribution is required (new strings and hence false positives will occur in the tail of the distribution). This gives

$$\begin{aligned} P(\varepsilon^+) &\approx \sum_{t=m(T)+1}^{m(\infty)} \left(p(s_t) - Tp(s_t)^2 + \frac{T^2}{2} p(s_t)^3 \right) \\ &\approx \sum_{t=m(T)}^{m(\infty)} \left(\frac{a}{i^b} - \frac{Ta^2}{i^{2b}} + \frac{T^2 a^3}{2i^{3b}} \right) \end{aligned}$$

Approximating the summation with an integral and setting $b = 2$,

$$P(\varepsilon^+) \approx \int_{m(T)}^{m(\infty)} \left(\frac{a}{i^2} - \frac{Ta^2}{i^4} + \frac{T^2 a^3}{2i^6} \right) di$$

$$\approx \quad \left(-\frac{a}{i} + \frac{Ta^2}{3i^3} - \frac{T^2a^3}{10i^5}\right)\Bigg|_{m(T)}^{m(\infty)} \tag{4.5}$$

As a first approximation, it is assumed that $m(\infty)$ is large enough so that $a/m(\infty) \approx 0$, then

$$P(\varepsilon^+) \approx \frac{a}{m(T)}\left(1 - \frac{Ta}{3m^2(T)} + \frac{T^2a^2}{10m^4(T)}\right)$$

The self test set, S_{test}, and the training set S_{trn} are contiguous, so $T = 1270000$ (the length of the training trace), and $m(T) = |S_{trn}| = 3763$. With $a = 5.54$, $P(\varepsilon^+) \approx 0.00147(1 - 0.1651 + 0.0245) = 0.00126$. Note that the error rate is a predictor of the false positive probability over a sufficiently long period of time, so the predicted false positive rate (assuming all new self strings are detected) for S_{test} is $\Delta\varepsilon^+ = 0.00126 \times 182000/7 = 33$ per day. The number of unique strings seen in S_{test} is 137, which gives $\Delta\varepsilon^+ = 137/7 = 19.6$, assuming all these strings are detected. Hence, the predicted false positive rate is in the same order of magnitude, but higher than empirical results. One of the sources of error is the assumption that the size of the self set is large, that is, $a/m(\infty) \approx 0$. The experimental value can be used to estimate the probability of a false positive as $P(\varepsilon^+) = 0.00075$; this can be used with equation 4.5 to determine a value for $m(\infty)$,

$$0.00075 \approx 0.00126 - \frac{a}{m(\infty)}\left(1 - \frac{Ta}{3m^2(\infty)} + \frac{T^2a^2}{10m^4(\infty)}\right)$$

Neglecting the higher order terms,

$$0.00075 \approx 0.00126 - \frac{a}{m(\infty)}$$

So $m(\infty) \approx 10900$, which is an estimate of the size of the self set, $|S_B|$. From this it is possible to estimate, that at a rate of 0.00075, it will take 9 million iterations to sample every self string at least once. This translates to 300 days, so for all practical purposes it is likely that there will always be false positives occurring, although the error rate will be dropping. For this reason, it is essential to have mechanisms such as activation thresholds and sensitivity levels to reduce false positives.

Figure 4.11 shows the effects of activation thresholds and sensitivity levels on $\Delta\varepsilon^+$. In this figure, only the activation threshold is varied; the sensitivity, ω, and the decay factors, γ_ω and γ_{match} are all set to the default values. If the detection system produced one false positive per unique new self string in S_{test}, then the false positive rate per day would be $\Delta\varepsilon^+ = 17.6$. With a threshold of $\tau = 1$, it is higher, at 70 per day. This indicates that the new self strings occur multiple times, and counting repeats, there are 78 new strings per day. To reduce false positives, the detection system should have some mechanism whereby new self strings only cause false alarms a single time, that is, the first time they appear. Such a mechanism is described in the next chapter (5).

The use of activation thresholds reduces $\Delta\varepsilon^+$, for example, when $\tau = 10$, $\Delta\varepsilon^+$ drops from 70 to 7.8. Note that the effects of activation thresholds can be predicted by assuming that false positives are reduced in proportion to the threshold, that is,

$$\Delta\varepsilon_\tau^+ = \frac{\Delta\varepsilon_1^+}{\tau} \tag{4.6}$$

where $\Delta\varepsilon_\tau^+$ is the false positive error rate for a threshold of τ.

So, for $\tau = 10$, the predicted false positive rate would be $\Delta\varepsilon_{10}^+ = 70/10 = 7.0$, which is close to the actual value of 7.8.

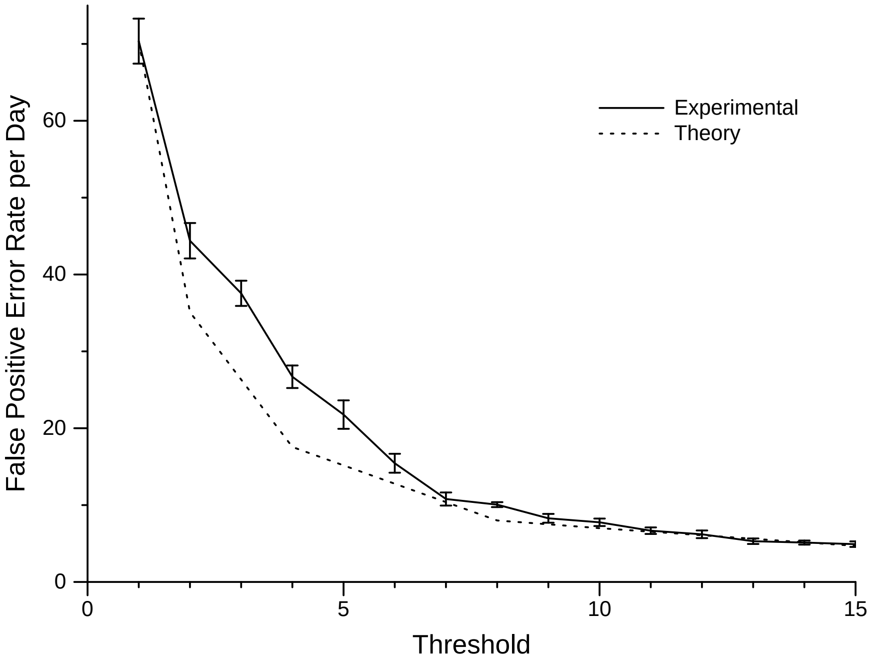

Figure 4.11: The effect of activation thresholds on false positive rates for S_{test} (τ varies). The y-axis gives the number of false positives per day against S_{test}. The *Theory* curve is derived from equation 4.6.

4.3.5 Detecting Real Nonself

The performance of the system on real nonself test sets is described here. Table 4.4 summarizes the detection rates against the real test sets for $\tau = 1$ and $\tau = 10$. For all these results the sensitivity was set to the default, although sensitivity only makes a difference for the SI test set; for the others it has no effect (data not shown). This table shows the detection rate computed over the whole trace, including self strings, ψ_{all}, and it shows the detection rate computed over only the nonself strings, $\psi_{nonself}$.

With $\tau = 1$, for all test sets $\psi_{nonself} = 1$, indicating that there were no false negative errors. When the threshold is increased to $\tau = 10$, all incidents are still detected, assuming an incident is detected when $\psi_{nonself} > 0$. The higher the value of *dr* the better, because high ψ values emphasize the separation between intrusive incidents and false positives. Even with $\tau = 10$, the average detection rate is high (0.81).

Figure 4.12 shows how changing the number of detectors affects the detection rate when using a threshold of $\tau = 10$. The lower curve plotted in the figure is the average ψ_{all} over all test sets, $\langle\psi_{all}\rangle$. The horizontal dashed line indicates the maximum possible $\langle\psi_{all}\rangle$, which is determined by the fraction of nonself strings in the traces. The average detection rate is approaching the maximum possible, but there are diminishing returns: as more detectors are added decreasing increments in performance are gained; at $n_d = 400$, $\langle\psi_{all}\rangle = 0.50$, as compared to 0.46 for $n_d = 100$. The upper line indicates the fraction of incidents detected, that is, the fraction of incidents for which $\psi_{all} > 0$. For $n_d > 40$, all seven incidents are detected. Although it is true that it suffices to detect a few nonself strings to detect these incidents, as mentioned before, the higher the detection rates, the clearer the separation between incidents and false positives. Furthermore, higher detection rates improve robustness: the higher the detection rates, the less damaging the compromise of a few detection systems.

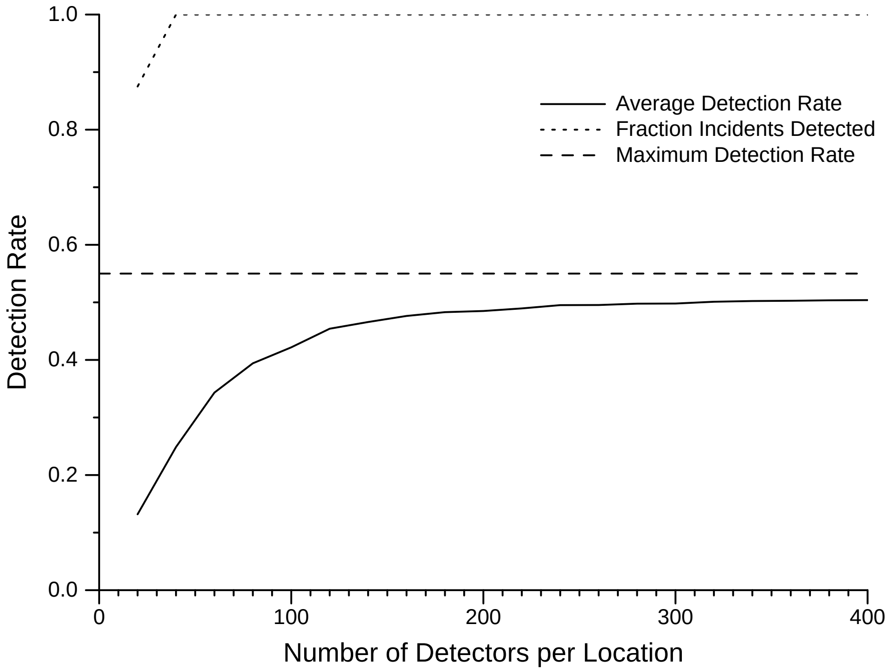

Figure 4.12: How the number of detectors impacts on detection rate (n_d varies, $\tau = 10$). *Fraction Incidents Detected* denotes the fraction of incidents detected, that is, the fraction of incidents for which $\psi_{all} > 0$. The *Maximum Detection Rate* is the fraction of nonself strings in each trace, averaged over all test sets, and *Average Detection Rate* is the average ψ_{all} over all test sets, $\langle\psi_{all}\rangle$.

Test Set	$\tau = 1$		$\tau = 10$	
	ψ_{all}	$\psi_{nonself}$	ψ_{all}	$\psi_{nonself}$
AP	0.54±0.00	1.00	0.48±0.01	0.89
PS	0.44±0.00	1.00	0.41±0.01	0.93
LP1	0.62±0.00	1.00	0.59±0.01	0.95
LP2	1.00±0.00	1.00	0.86±0.01	0.86
LP3	0.10±0.00	1.00	0.02±0.00	0.85
SP1	1.00±0.00	1.00	0.56±0.05	0.56
SP2	0.17±0.00	1.00	0.11±0.01	0.65
average	0.55	1.00	0.43	0.81

Table 4.4: Detection rates against real test sets (τ varies). ψ_{all} is the fraction of anomalous strings in the complete trace, including self strings, and $\psi_{nonself}$ is the fraction of nonself strings in the trace that are classified as anomalous.

ID systems are usually evaluated in terms of a Receiver Operating Characteristics (ROC) curve, which indicates the trade-off between false positives and detection rates. The ROC curve for this system is shown in figure 4.13. This plot is derived by varying the activation threshold, although there are other parameters that can be varied to effect this trade-off. The detection rate is $\langle \psi_{all} \rangle$ averaged over all seven real test sets. On an ROC curve, the best system is the one which has performance closest to the top left-hand corner of the plot. The default threshold that that is used, $\tau = 10$, is close to this corner. However, the actual threshold chosen will depend on the application. If detection of nonself is critical and false positives can be tolerated, then a lower threshold will be more suitable. If, however, false positives are completely undesirable, then a higher threshold will be preferable. Note that decreasing the threshold below 7 makes little difference for detection rates *on this suite of test sets*, and increasing it beyond 13 makes little difference for false positive rates. However, the false positive rate can always be reduced to zero with a large enough threshold; in the extreme case, if τ is the length of the trace there can be no false positives, and no detection either.

4.3.6 Increasing the Size of the Self Set

The impact of the size of the LAN being monitored on the false positive rate was tested by collecting network traffic from a different subnet at the Computer Science department at UNM. This subnet was larger than the original subnet, with 72 computers instead of 50; both data sets were combined for a total of 120 computers. For these 120 computers, there were 7500 unique strings, representing 7500 unique datapaths, which means that the additional subnet contributed 3700 new unique strings.

Figure 4.14 shows the probability distribution for the 120 computer self set, S_{120}. The power-law fitted to the data has the same exponent as before ($b = 2$), but has a different offset, $a = 10$. This offset is roughly double that for the 50 computer set, S_{50}, which has half the number of unique self strings. If the number of strings in the sample self set is doubled, this will result in a "doubling" of the distribution, providing the added strings have the same power-law exponent. The consequence of "doubling" the distribution is a doubling of a. If the size of the sample self set is doubled and the exponent is unchanged, the power-law, equation 4.1, becomes

$$p_2(s_i) = \frac{a}{2}\left(\frac{i}{2}\right)^{-2}$$

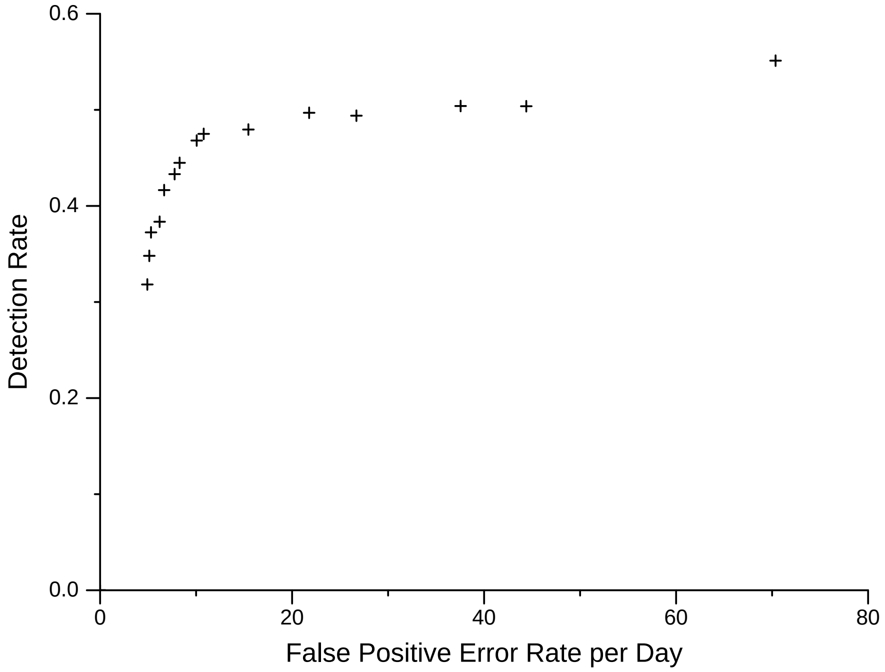

Figure 4.13: ROC curve for this system (τ varies). Each point corresponds to a different threshold, from $\tau = 1$ on the extreme upper right, to $\tau = 15$ on the lower left. The detection rate is the average of ψ_{all} over all 7 test sets.

$$\log p_2(s_i) \quad = \quad \log\left(\frac{a}{2}\left(\frac{1}{2}\right)^{-2}\right) - 2\log i$$

where p_2 is the “doubled” probability distribution. Let

$$a_2 = \frac{a}{2}\left(\frac{1}{2}\right)^{-2} \tag{4.7}$$

then

$$p_2(s_i) = a_2 i^{-2}$$

So a_2 is the offset for p_2, and $a_2 = 2a$, from equation 4.7. Thus the offset should double. In general, if the number of strings increases by a ratio R, then the offset should be multiplied by R.

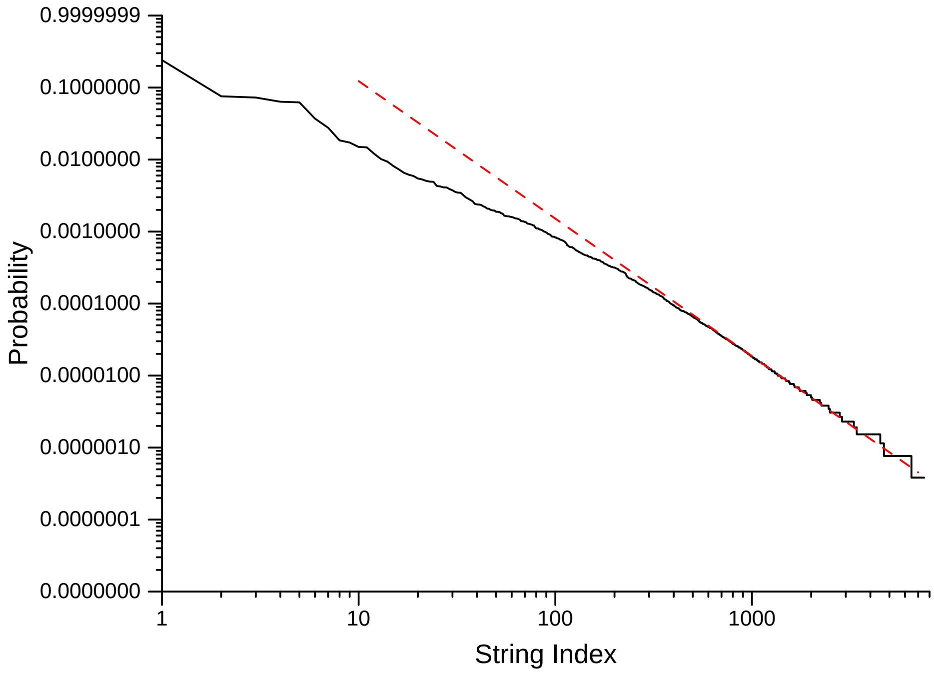

Figure 4.14: Sample distribution of self strings for 120 computers. The dotted line indicates the power-law, $p(s_i) = 10i^{-2}$. Note that both axes are logarithmic.

The predicted false positive rate for S_{120} is 0.0019, assuming that $a/m(\infty) \approx 0$, which is 50% greater than that for S_{50}. So doubling the size of the sample self set (by combining the two separate subnets) has not doubled the false positive rate. In general, if the exponent stays the same, increasing the size of the self set should increase the false positive rate, so that the probability of a false positive, given an increase in size determined by a ratio R, is

$$\begin{aligned} P_R(\varepsilon^+) &\approx \frac{Ra}{Rm(T)}\left(1 - \frac{TRa}{3(Rm(T))^2} + \frac{T^2(Ra)^2}{10(Rm(T))^4}\right) \\ &\approx \frac{a}{m(T)}\left(1 - \frac{Ta}{3Rm^2(T)} + \frac{T^2a^2}{10R^2m^4(T)}\right) \end{aligned}$$

because $a_R = Ra$, and $m_R(T) = Rm(T)$.

This equation indicates that the growth of the false positive rate is sub-linear in the growth of the self set. This is not scalable as the number of computers increases, but scalability was defined in terms of a fixed problem size in section 3.1.5. The problem of scalability in this context is defined by the question: given a self set generated by k computers, will running detection systems on k_1 computers be scalable to running detection systems on k_2 computers, where $k_1 < k_2 \leq k$? This is the case for false negative errors, because increasing the number of detection systems will increase the total number of detectors and hence decrease the false negative rate. This system does not scale in terms of false positives, but, as was shown in section 3.1.5, *any* system that produces false positives will not scale.

4.4 Summary

This chapter described a network ID system that is based on NSM, but differs as follows: it does not monitor traffic frequencies; only TCP SYN packets are monitored; services are grouped into a few well-known services; and some services (such as WWW servers) are filtered. It is not clear how these changes affect performance over that of NSM, because there are no points of comparison. There are no published results of the performance of NSM (in terms of either false positives or false negatives) on data sets that allow for replication of experiments.

Two mechanisms were introduced for reducing false positives: activation thresholds and sensitivity levels. These mechanisms are based on the assumptions that nonself strings occur in temporal clumps relative to new self strings, and that intrusive incidents are characterized by multiple nonself strings (connections). This assumption holds for the self and nonself sets collected on a real system and used in the experiments. Activation thresholds and sensitivity levels reduce false positive rates, and so can be used to adjust the trade-off between false negatives and detection rates. Generally, false positive rates are reduced by a factor of τ, where τ is the activation threshold. However, detection rates are reduced less, because the nonself strings are clumped, for example, the detection rate averaged over all real nonself incidents is reduced from 100% at $\tau = 1$, to 81% at $\tau = 10$. Consequently, activation thresholds provide a mechanism for increasing the separation between detection of nonself and false positive errors.

The distribution of the less frequently occurring self strings in the real sample is accurately modeled by a power-law, which can be used to model the self strings as a strongly stationary random process. This, together with the theory developed in chapter 3, makes it possible to predict false positive rates. Because of this long-tailed distribution, it is unlikely that all the self strings can be collected in a practical period of time, so new self strings should always be expected to occur, albeit at a slow rate. A power-law distribution with the same exponent also holds for a different LAN, one that consists of 50% more computers. However, the offset differs; it is proportional to the number of strings distributed according to a power-law with the same exponent. So doubling the number of self strings doubles the offset. Increasing the size of the self set also increases the number of false positives; in these experiments doubling the self set resulted in a 50% increase in the false positive rate.

When the nonself test set is near to the self set (measured in Hamming space), having multiple secondary representations improves detection rates, by up to a factor of three. When the nonself test set is randomly distributed with respect to the self set, multiple secondary representations make no difference, except that in the case of the linear congruential implementation of imperfect hashing, the detection rates are reduced, because imperfect hashing introduces new holes. On all test sets, the contiguous bits match rule together with substring hashing works best in terms of the trade-off between detection rates and the expense of detector generation. It is important that the secondary representation does not spread the self strings too much, because the number of retries for detector generation is less with a structured or clumped self set than for a random self set (as much as three orders of magnitude less).

All local detection systems contribute similarly to the global detection rate; there is no single secondary representation that is significantly better than any others. This indicates the need for multiple representations, but it also implies that the distributed detection system is robust because loss of one or a few locations will result only in slight reductions in detection rates. With only 100 detectors per location on 50 computers (simulated), all 7 nonself incidents are clearly detected (56 to 100% of nonself strings detected), even with activation thresholds high enough to reduce the false positive rates to 8 per day.

Chapter 5

Extensions to the Basic Model

This chapter describes the implementation and testing of mechanisms to achieve three goals:

Elimination of autoreactive detectors: If a detector is activated by a self string (that is, it is autoreactive), then it should be eliminated as soon as it becomes clear that a false alarm has been raised. This will reduce the false positive rate because the same self string will not trigger the same detector multiple times.

Adaptation to changing self sets: The self set can change, for example when new computers are added, users change their habits, new software and hardware are installed, etc. All these changes can lead to false positives, so the system must adapt to the changing self sets. It must do this in a way that is automated, distributed, and resistant to subversion.

Signature-based detection: For known patterns of intrusions, signature-based detection can be more effective than anomaly detection, with reduced false positive rates and higher detection rates. To improve protection, every detection system should perform anomaly detection, and once an intrusion is detected, encode the characteristic patterns associated with that intrusion as a signature, so that in future that anomaly can be responded to more rapidly and with less error. Other ID systems use signature-based detection, but the signatures are usually crafted by a human with prior knowledge; they are not automatically extracted from the anomalies that have been detected. The Immune System (IS), by contrast, has mechanisms that allow it to automatically extract the signatures of anomalies and store those to facilitate future responses.

There are four mechanisms that have been used to achieve these goals: costimulation, distributed tolerization, dynamic detectors, and memory detectors. These mechanisms will be explained in detail in this chapter.

5.1 The Mechanisms

This section describes the implementation of the four mechanisms. Because these mechanisms increase the complexity of the system, little theory is presented for predicting performance. However, the simple theory that is reported explains some of the the effects of the mechanisms, and the empirical results reported in section 5.2.

5.1.1 Costimulation

Costimulation in the immune system provides a way of eliminating autoreactive lymphocytes. Section 2.1.4 described how every lymphocyte requires two signals to be activated: signal I is provided by binding to pathogen epitopes, and signal II is provided by other cells of the immune system (see section 2.1.4). If signal I is received in the absence of signal II, the lymphocyte dies or becomes anergic. The second signal is a way of confirming that nonself was indeed detected. In the immune system signal II for Th-cells is thought to be provided in the presence of tissue damage, so this suggests a damage indicator for an artificial system.

Similar mechanisms are used here, where signal I is provided by matching τ strings and being activated, and signal II is a damage signal provided by a human operator. Whenever a detector d is activated, it raises an alarm, and that alarm is communicated to some location where it can be observed by a human operator (for example, as an email message). If the human operator determines that the alarm is indeed caused by nonself, then he or she sends signal II to d. If d does not receive signal II within a period of time T_s, called the costimulation delay, then d will "die", that is, it will be removed from the detector set (the s in T_s stands for *signal*). The costimulation delay is intended to give a human operator time to react to anomalies, so, typically it is set to 1 day. The default is that the detection is incorrect, and the human operator sends a signal to counteract this default, so this is a False Positive Default (FPD) scheme.

An alternative way of implementing costimulation would be for the default to be that detection is correct, and the human operator only sends a signal if the detector is autoreactive; this is a True Positive Default (TPD) scheme. Then any detector receiving signal II would die, otherwise it would not be affected. This scheme has two disadvantages. Firstly, false positives would require action on the part of the human operator, whereas with a FPD scheme, the human operator can ignore false positives and the system will automatically correct the problem. Because false alarms are undesirable, it is important to minimise the amount of human effort required to deal with false alarms. Secondly, TPD is vulnerable to subversion. If the system were corrupted so that an adversary was in the position of the human operator, then that adversary could send signals to all detectors telling them to die, thus disabling the detection system completely. With TPD, the worst that such an adversary could do is *not* confirm autoreactive detectors, which could only result in higher false positive rates.

Ideally, a human operator should not be needed; the system should have some notion of "damage" on which to base costimulation. This would require automated response, an issue which is discussed later in chapter 6.

5.1.2 Distributed Tolerization

The previous section described how autoreactive detectors can be eliminated from detector sets. If the self set changes, detectors will be continually eliminated through costimulation, and the number of detectors will drop to zero over time, with a corresponding drop in detection rates. What is needed is a way of generating new detectors on-line to replace those which have been eliminated. One way of doing this would be a Central Tolerization (CT) scheme: a single location l_{thymus} would have an accurate generalization of the self set, and detectors would be generated using the negative selection algorithm at l_{thymus} and then distributed to other locations as necessary. This is similar to CT in the immune system, which takes place in the thymus, hence the subscript for the location (see section 2.1.4). CT was used for the off-line generation of detectors in chapter 4.

There are several problems with CT. Firstly, having a single central location for tolerization means the system is not robust: compromise of that location will disable the system. Secondly, the more accurate the generalization of the self set, the more information required for storage of this generalization; this information could exceed the storage capacity of any one location. This is not the case for the ID system developed here, but in general it cannot be assumed that there will be enough capacity at any one location. Finally, CT requires communication: when a local detection system needs a new detector it must communicate with l_{thymus}, which must then reply with the information concerning the new detector. As the number of local detection systems needing detectors increases, and the frequency with which detectors are needed increases, so the communication costs increase. It is possible that for certain applications these cost could make the ID system impractical.

For these reasons, a method for Distributed Tolerization (DT) has been implemented. One way of generating a new valid detector d' at location l would be to tolerize it using the information stored in the existing detector set, D_l. However, this is pointless. Recall from section 3.2.2 that a valid or tolerized detector d, is one for which the complement of its cover is a generalization of the self set, that is, $S \subset C_d^C$. So, for a set of detectors D_l,

$$S \subseteq \bigcap_{d \in D_l} C_d^C \Rightarrow S \subseteq U - \bigcup_{d \in D_l} C_d$$

A new detector d' will only be valid if $S \subseteq C_{d'}^C$, but the only information available about self at location l is from the detector set D_l, so we require that

$$\begin{aligned} U - \bigcup_{d \in D_l} C_d &\subseteq C_{d'}^C \\ &\subseteq U - C_{d'} \end{aligned}$$

which means $C_{d'} \subseteq \bigcup_{d \in D_l} C_d$. Hence, every string matched by d' must also be matched by some detector $d \in D_l$. So d' is useless because it adds nothing to the detection.

What is required is some additional source of information regarding the self set. If it is assumed that self strings occur more frequently than nonself, then this information is provided by the random process that produces the self strings, that is, it is provided by the system being monitored. This information is used to implement DT as follows. When a new detector, d', is created, it consists of a randomly generated bit string. It is initially in an immature state, which means that if it matches *anything once*, it does not signal an anomaly (raise an alarm), but is eliminated and replaced (regenerated)[1]. This immature state lasts for T time steps; T is called the tolerization period. If d' survives the tolerization period (it does not match anything for T time steps), it becomes mature. When mature, d' functions exactly as an ordinary or naive detector: it only becomes activated after accumulating $\tau - \sigma_l$ matches (where σ_l is the current sensitivity level at location l), and it will die T_s time steps after activation if it does not receive costimulation. With this implementation of Distributed Tolerization (DT), no communication is needed, but during tolerization, immature detectors are consuming resources and not contributing to the detection of nonself.

There is a trade-off between the fraction of immature detectors, ι, and the false positive rate, which is governed by the match decay, γ_{match}, and the tolerization period, T. The accumulation of matches can be modeled as a queue [Williams, 1991, pages 215–217], where the time between arrivals is modeled by an exponential random variable, with parameter λ_A (the arrival rate), and the time between departures is also a random exponential variable, with parameter λ_D (the departure rate). If $\lambda_A > \lambda_D$, then there is no stationary distribution for the number of matches in the queue, and the queue could grow indefinitely. This means that any match will contribute to activation, and over a sufficiently long period of time, every new self string will cause a false positive. Conversely, if $\lambda_A < \lambda_D$, then the queue is *stable*, and is described by a unique stationary distribution. In this case, the queue will not grow indefinitely, which means that only some new self strings will cause false positives.

[1] This is a temporal version of the negative selection algorithm.

The probability that the queue and hence the number of matches exceeds the threshold can be computed as follows. Let the random variable Y be the number of matches in the queue, then the probability that there are n matches in the queue at any time is given by [Williams, 1991]

$$P(Y = n) = R^n(1 - R)$$

where $R = \lambda_A/\lambda_D$ is the ratio of arrivals to departures. So, the probability that the number of matches exceeds the threshold τ, is

$$\begin{aligned}
P(Y \geq \tau) &= 1 - P(Y < \tau) \\
&= 1 - \sum_{j=0}^{\tau-1} P(Y = j) \\
&= 1 - \sum_{j=0}^{\tau-1} R^j(1 - R) \\
&= 1 - (1 - R)R\left(\frac{1 - R^{\tau-1}}{1 - R}\right) - (1 - R) \\
&= R^\tau \qquad (5.1)
\end{aligned}$$

The queue of matches for a detector d has an arrival rate which is determined by the probabilities of occurrence of the strings matched by d, that is by all $s_i \in C_d$, and a departure rate given by the probability of match decay, $\lambda_D = \gamma_{match}$. Now if the probability of occurrence of a string s_i is $p_i \geq 1/T$, then s_i is expected to occur at least once during the tolerization period T. Hence, it is expected that d is tolerized to s_i, and will not be activated by s_i; only strings for which $p_i < 1/T$ are expected to activate d. Assuming that there is only one such string, s_i that d matches, then the arrival rate for the match queue of d is given by $\lambda_A = p_i$. Substituting in equation 5.1, and expressing the probability of match decay as a decay period, t, gives

$$P(Y \geq \tau) = \left(\frac{p_i}{1/t}\right)^\tau$$

Because $p_i < 1/T$, this becomes

$$P(Y \geq \tau) < \left(\frac{t}{T}\right)^\tau \qquad (5.2)$$

When the decay period, t, is close to the tolerization period, T, the activation threshold has less effect on the probability of activation, and as the T increases relative to t, the activation threshold has an increasing effect. But note that, for practical purposes, if τ is moderate, then once the tolerization period is even moderately larger than the decay period, the probability of activation is so low that increasing the tolerization period further has a negligible effect. For example, consider the case where the decay period is $t = 1$ day, and the threshold is $\tau = 10$, then for $T = 7$ days, $P(Y \geq \tau) < 3.5 \times 10^{-9}$, which is close enough to zero that an increase in T will have no practical effect. So, in practice, it is expected that changes in the tolerization period will only have an effect on false positive rates if the tolerization period is similar to the decay period.

According to this analysis, if $T \leq t$, then $P(Y \geq \tau) = 1$, which means that every new self string will be a false positive every time it occurs. However, this is not the case, because the match queue dynamics do not agree with this model: not only is an element in the queue removed according to the departure rate, but all elements are removed whenever the queue reaches a size of τ. This means that the queue cannot grow indefinitely, and the activation threshold should still reduce the false positive rate, in the simplest case by a factor of roughly τ, as shown in section 4.3.4. In fact, in this case there is no unique stationary distribution for the number of matches (the length of the queue).

5.1.3 Dynamic Detectors

If coverage of nonself by the detection system is incomplete, then an adversary could discover and exploit the gaps to evade detection. The longer the period of time over which the coverage remains unchanged, the more likely it is that an adversary will find gaps, and once found, those gaps can be exploited more often. The replacement of autoreactive detectors will change the coverage, but if false positive rates are low, then the change in coverage will be correspondingly slow. One solution is to increase the number of detectors so that there are no gaps, but when resources are limited this may not be possible. An alternative solution is to change the detectors over time, that is, to make them dynamic. This can be done by giving each detector a finite lifetime, measured in time-steps. At the end of this lifetime, the detector is eliminated and replaced by a new randomly-generated detector that is tolerized using the DT scheme described in the previous section.

If all the detectors have the same lifetime, and they begin life at the same time-step (for example, at the start of monitoring), then having a fixed lifetime will mean a complete loss of detection when all detectors die at the same time-step, and are replaced by new immature detectors. So detectors are given a probabilistic lifetime, defined by a parameter p_{death}, which is the probability that a detector will die at any given time-step. Using a constant probability means that the lifetime of a detector is a geometric random variable, with an expected value of $1/p_{death}$. With a probabilistic lifetime, death events should be more evenly distributed over time[2].

A limitation is imposed on detection rates by dynamic detectors: detectors must remain immature for some period of time, during which they will not be contributing to the detection rate. The expected fraction of immature detectors, $E(\iota)$, is given by

$$E(\iota) = \frac{p_{death}}{1/T} = Tp_{death} \tag{5.3}$$

Equation 5.3 implies that $p_{death} \leq 1/T$, that is, the expected lifetime, $E(\text{life}) = 1/p_{death}$, must be greater than the tolerization period. The tolerization period, T and the lifetime, $E(\text{life})$, must be adjusted to give detectors sufficient time to mature and be useful. The closer $E(\text{life})$ is to T, the more detectors will be immature, but the further it is, the more attackers will have time to discover and repeatedly exploit particular vulnerabilities in the coverage. This trade-off depends on how long it takes an attacker to find and exploit a vulnerability, over what period of time the attacker is likely to repeatedly exploit the same vulnerabilities, and on how fast the self set is changing.

5.1.4 Memory

Although dynamic detectors protect against repeated exploitation of vulnerabilities, dynamic detectors alone will not improve average detection rates. If the nonself test set, N_{test}, is new or novel, the only way to improve detection would be to increase the number of detectors and/or change the specificity of the match rule. However, if N_{test} has been encountered before, then the detection rate can be improved by retaining a "memory" of N_{test} and using this for subsequent detections; that is, by using signature-based detection. Information about previous nonself is stored in the form of memory detectors. A memory detector is an ordinary detector with certain additional properties:

[2] An alternative would be to initialize the detectors to die asynchronously.

Unlimited lifetime: A memory detector has no probability of dying. For memory detectors, $p_{death} = 0$.

Heightened sensitivity: A memory detector will signal an anomaly after a single successful match. For memory detectors, $\tau = 1$.

Memory detectors are selected as follows. Whenever a naive (non-memory) detector is activated by a string s, it enters into a competition to become a memory detector. If several other detectors have also been activated by s, the detector d_{best} with the best match is chosen to become a memory detector. For the contiguous bits rule, this is the detector which matches s in the most contiguous bits. d_{best} does not immediately become a memory detector, but is a provisional memory detector until it receives costimulation, confirming that the match was nonself. At that time it becomes a full memory detector.

There is a limit, m_d, to the number of memory detectors that can exist in any location. This limit is expressed as a fraction of the total number of detectors at the location, so the number of allowed memory detectors is given by $m_d n_d$. This limitation is intended to prevent all naive detectors from becoming memory detectors, which could mean a loss of anomaly detection capability. If a provisional memory detector, d_{prov} receives costimulation, and the number of memory detectors is less than $m_d n_d$, d_{prov} will become a memory detector. However, if the number of memory detectors is equal to $m_d n_d$, then an existing memory detector must be removed to make d_{prov} into a memory detector. The detector to be removed is selected randomly from amongst the $m_d n_d$ existing detectors.

Memory detectors alone will not improve subsequent detection of nonself, except for the fact that memory detectors have reduced activation thresholds. If a string $a \in N$ was never detected previously by a memory detector, there is no reason why it will be detected by one later. For this, dynamic detectors are essential, because they provide the changing coverage to detect a, while the memory detectors ensure that what has been detected in the past will still be detected now. Together, memory detectors and dynamic detectors improve detection rates, as well as eliminate consistent vulnerabilities and improve responses to previously encountered nonself.

5.1.5 Architectural Summary

To summarize, the architecture of the distributed ID system is shown in figure 5.1. All internal computers (those on the LAN) are assumed to run local detection systems. These local detection systems all monitor the same traffic because of the broadcast assumption. The only aspect of traffic that is monitored are the datapaths formed by TCP SYN packets, which are represented as 49-bit strings, using the base representation (see section 4.1.1). Each local detection system has the following components:

Sensitivity level: This indicates the current sensitivity value that must be subtracted from the threshold, τ, for activation. See section 4.1.3.

Secondary representation parameters: These are the parameters necessary for implementing the secondary representation. Each local detection system has different, randomly determined values for these parameters. In the case of the default secondary representation, substring hashing, there are two parameters: a 256-byte hash table, and a permutation mask of 6 digits. See section 4.1.2.

Detector set: The detector set consists of a fixed number of detectors. See section 4.1. Detectors are generated off-line using the negative selection algorithm (see section 3.2.2), or on-line using DT (see section 5.1.2).

Each detector in a detector set has the following components:

State: This can be immature, naive or memory. Immature detectors are described in section 5.1.2, memory detectors are described in section 5.1.4, and naive detectors are detectors that are neither memory nor immature.

Activation flag: This indicates that the detector has been activated, and has yet to receive costimulation.

Last activation: This indicates the number of time-steps since the last activation. This, together with the activation flag, is used for costimulation. See section 5.1.1.

Number of matches: This is the match count, c_d, which keeps track of the number of times the detector has matched. See section 4.1.3.

Binary string: This is a 49-bit string which is used to encode compactly the generalization of the training set.

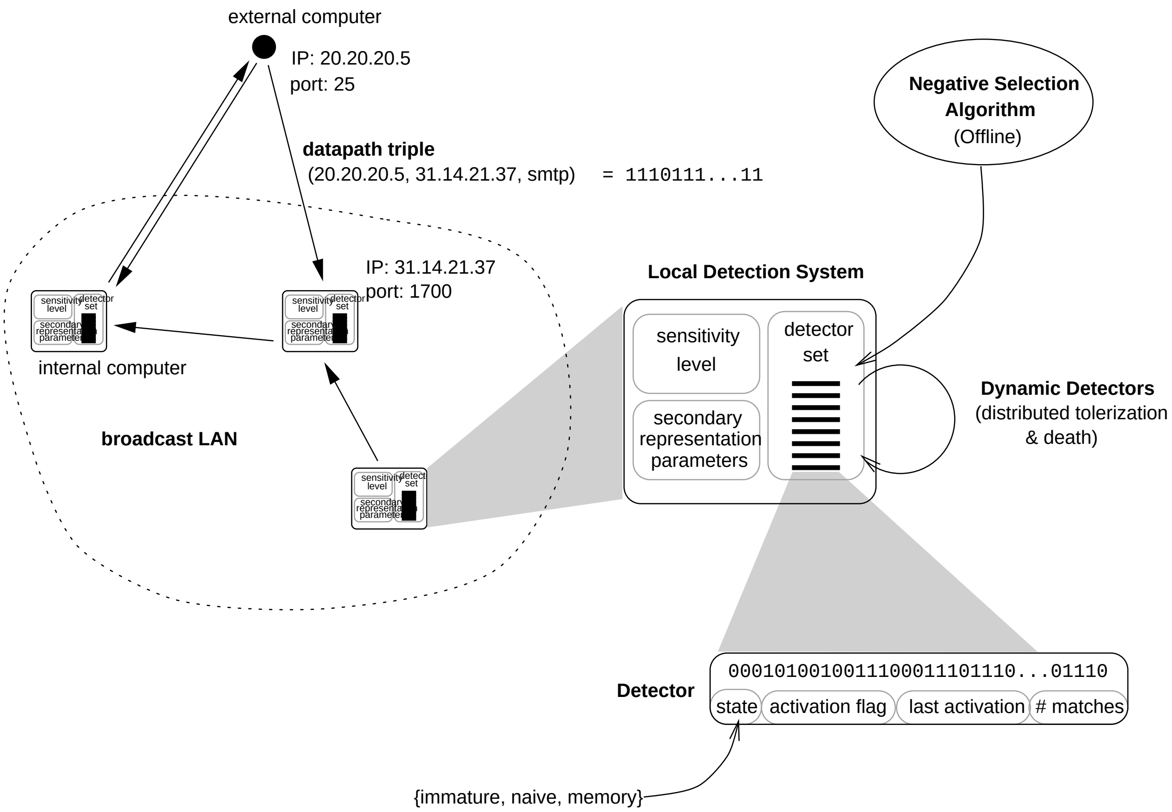

Figure 5.1: The architecture of the distributed ID system.

The lifecycle of a detector is given by figure 5.2. A detector consists of a randomly created bit string that is immature for the tolerization period (in this diagram, two days). If it matches anything once during that period it dies and is replaced by new randomly generated detectors. If it survives tolerization, it becomes a mature, naive detector, that lives for an expected $1/p_{death}$ time-steps (in this diagram seven days). If the detector accumulates enough matches to exceed the activation threshold, it is activated. If the activated detector does not receive costimulation, it dies. If it receives costimulation, it enters a competition to become a memory detector. Once a memory detector, it lives indefinitely, and only requires a single match for activation.

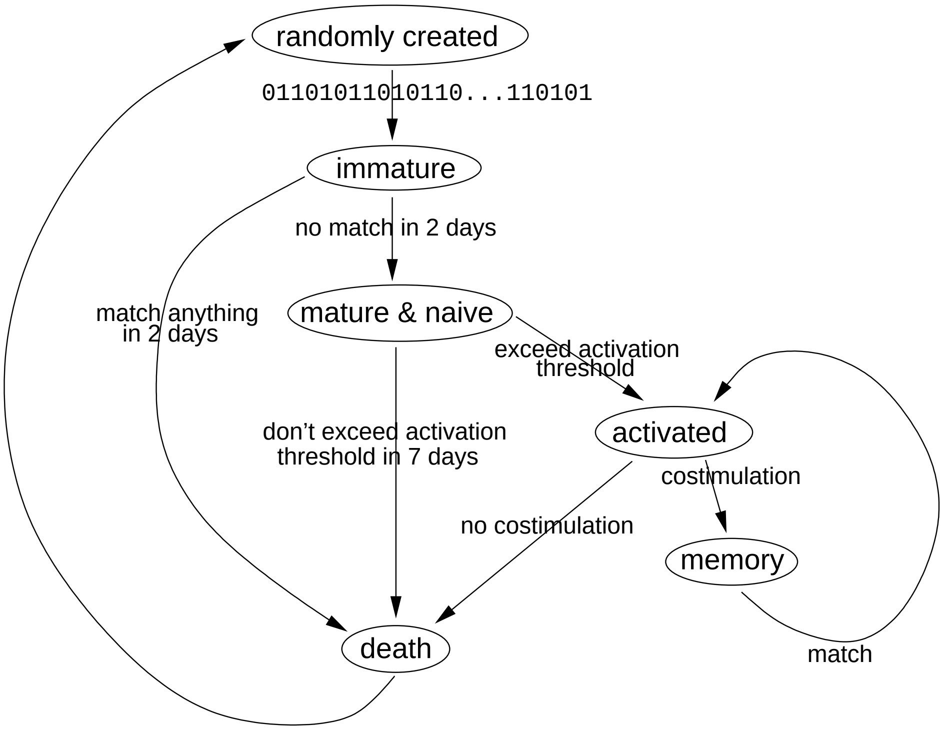

Figure 5.2: The lifecycle of a detector.

5.2 Experimental Results

The effect of these mechanisms was tested on the elimination of autoreactive detectors, on changing self sets, and on signature-based detection. For the first experiment (section 5.2.1), actual data were used, but for the last two experiments (sections 5.2.2 and 5.2.3), the self strings were produced by the simulation of a strongly stationary discrete random process, $X = \{X_1, X_2, \ldots\}$, where each random variable X_i is a function producing a self string $s \in S$ at time-step i. The random variables all have the same distribution, which is taken as the sample distribution of the real data collected for $S_{trn} \cup S_{test}$ (see section 4.2.1). So, at each time-step, a string $s \in S_{trn}$ is randomly selected from the distribution of $S_{trn} \cup S_{test}$. The actual probabilities from the sample are used, and not simply the power-law approximation to the distribution.

Simulation increases the flexibility in the way the experiments can be set up and run; in particular, there is no limit on the number of time-steps for which the simulation can be run. This enable periods of months to be simulated. Figure 5.3 compares the distribution of the simulated self strings, with the distribution of the actual self strings, over the same time period. The curves are identical except at low probabilities (0.00001). It was assumed that the self set is generated by a strongly stationary process.

The parameters for the full distributed detection system are given in table 5.1. As in the previous

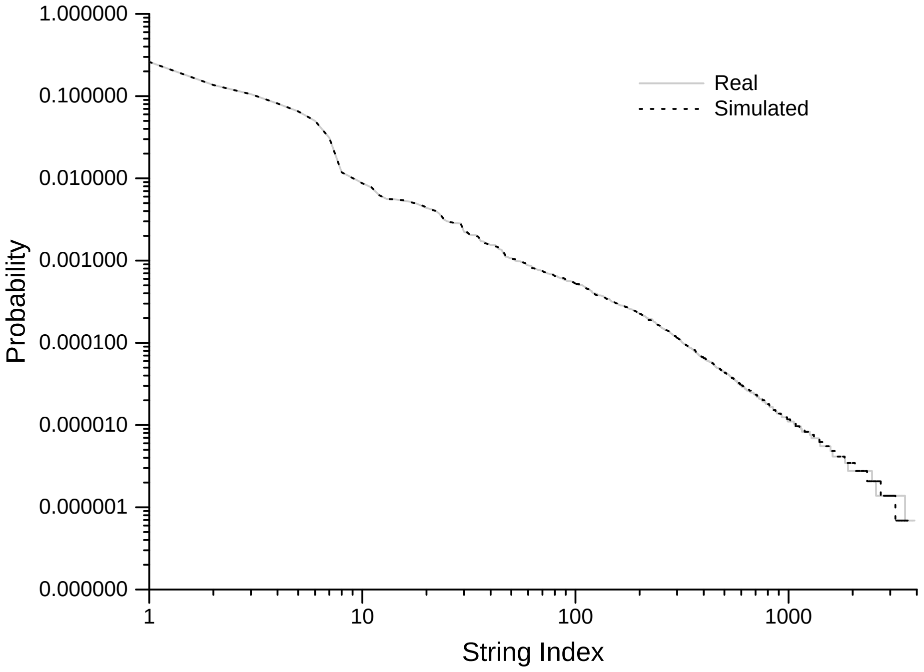

Figure 5.3: The probability distributions for real and simulated self. Note that both axes are logarithmic.

chapter, the default values given in the table are used throughout these experiments, unless otherwise stated. Parameter values that differ from the default are given in the captions for every table and figure. Where multiple runs with different random seeds are performed, the usual number is 30, unless otherwise stated. The results of multiple runs are reported as a mean, followed by a 90% confidence interval after a $\pm$ sign, unless otherwise stated. Elapsed time for the experiments is reported in *days*. For all simulated data, one day is assumed to be equivalent to 25000 time-steps, that is, 25000 TCP SYN datapaths. Where the fraction of immature detectors, ι, is reported, this fraction is measured over all detector sets at all locations.

5.2.1 Costimulation

The goal of costimulation is to eliminate autoreactive detectors. Autoreactive detectors are detectors that incorrectly detect self datapaths and so cause false positives. Costimulation in this context is based on the immunological mechanism of costimulation, where a lymphocyte needs two disparate signals to be activated.

Costimulation was tested using the training data, S_{trn}, and the self test set, S_{test}, described in section 4.2.1. The test data were not simulated. The default parameter settings were used for the tolerization period, that is, $T = 14$ days, so no new detectors matured during the testing phase, because the test set only spans seven days and it was assumed that all detectors were mature at the start of the test phase. In an on-line system, a human operator would have to provide the second signal to detectors that send alarms. To automate these experiments, it has been assumed that for all detectors matching nonself strings (i.e. autoreactive detectors), a human operator confirms the anomalies and sends signal II to the detectors, within the costimulation delay period, T_s. Thus it has been assumed that the human operator is *prompt* and *correct*. In the experiments reported in this section, however, there are no nonself strings in the test set, so the effect of the costimulation is that all detectors that are activated will die after the costimulation delay has elapsed, that is, after T_s

Parameter	Description	Default Value
T_s	costimulation delay	25000 (1 day)
T	tolerization period	350000 (14 days)
m_d	maximum fraction memory detectors	0.0
p_{death}	death probability	0.0
n_L	number locations	50
n_d	number detectors per location	100
ℓ	string length	49
r	match length	12
Λ	secondary representation	substring hashing
h	match rule	contiguous bits
τ	activation threshold	10
γ_{match}	match decay	0.0
ω	sensitivity	1.5
γ_ω	sensitivity decay	10

Table 5.1: The parameters for the basic distributed ID system. Unless stated otherwise, all experiments use these default values. Note that when $\tau = 1$, the settings for γ_{match}, ω, and γ_ω are irrelevant, because τ can never be less than 1

time-steps.

The idea behind costimulation is to reduce false positive rates to those equivalent to the number of unique new self strings. As shown in table 5.2, this does not happen, because of the delay before the autoreactive detectors die, during which they can continue to trigger false alarms. The shorter the delay, the lower the false positive rate. The shortest delay shown in the table is $T_s = 0.0$ days, but actually the delay is 100 time-steps (the equivalent of about 6 minutes), which was assumed to be the minimum reasonable time for an operator to react, assuming the operator was present at a console immediately the signal was sent. For this shortest delay, the false positive rate comes close to the number of unique new self strings: when the activation threshold is one, the false positive rate is 30 per day, whereas the number of unique new self strings is 18 per day. There is no unique count for $\tau = 10$.

Costimulation reduces false positive rates. In the table, $T_s = \textit{infinite}$ indicates that there is no costimulation, so autoreactive detectors are not eliminated. In this case, the false positive rates are two to three times higher than the best false positive rates for costimulation. For a costimulation delay of $T_s = 1$ day, the false positive rate is halved from the case with no costimulation. ι, measured at the end of the seven days of test data, is less than or equal to 0.061 in all cases, indicating that no more than 305 out of 5000 detectors are immature at any given moment.

5.2.2 Changing Self Sets

The efficacy of distributed tolerization and costimulation as mechanisms for adapting to changing self sets was tested with a simulation of a massive change in the self set. The detectors were tolerized off-line on 7 days of data (the self test set, S_{test}), and then the simulation was run for 12 weeks of simulation time, or 2.1 million time-steps. During this test phase, the simulation of the random process X, produced self strings drawn from the distribution of the set of self strings, $S_{test} \cup S_{trn}$. So, whereas the detectors were tolerized against 869 self strings during the test phase, they were presented with 3900 self strings during the training phase. Because of computational constraints, each experiment was carried out for only 10 runs with different random seeds, instead of the usual 30 runs.

τ	T_s days	$\Delta\varepsilon^+$ per day	ι
1	unique count	17.6	n/a
1	0.0	30.0±1.5	0.061±0.001
1	1.0	52.0±2.0	0.039±0.001
1	infinite	70.3±2.9	n/a
10	0.0	2.6±0.2	0.007±0.000
10	1.0	4.3±0.3	0.004±0.000
10	infinite	7.8±0.5	n/a

Table 5.2: Costimulation results (τ varies, T_s varies). The *unique count* indicates the number of unique self strings in S_{test} that were not in S_{trn}, the costimulation period of 0.0 days ($T_s = 0.0$) was actually a costimulation period of 100 time-steps, and ι is the fraction of immature detectors at the end of the run.

Figure 5.4 shows the changing false positive rate of a typical run for each of five different tolerization periods, $T = \{1, 7, 14, 21, 28\}$ days. The false positive rates are high initially, over 1000 per day, but then drop exponentially (note that the y-axis is logarithmic); after a week the false positive rate has dropped to under 100 per day. The rates continue to drop, but more slowly. These results are to be expected because of the distribution of $S_{test} \cup S_{trn}$: 5% of the self strings occur 95% of the time, so the strings that would cause false positives most of the time tolerize the system rapidly. Note that the false positive rates are different for the different tolerization periods, as expected if there is no decay of the match counts. False positive rates averaged over the last 7 days of the simulation, over 10 runs, are 39 per day for $T = 1$ day, and 9.9 per day for $T = 28$ days (4 weeks). In the latter case, the false positive rate is still double that achieved against S_{test} in the previous section, where it was 4.3 (see table 5.2).

Although longer tolerization periods are preferable for reducing false positives, they will increase ι, and so reduce detection rates for a fixed number of detectors. Figure 5.5 shows the change in ι for the five different typical runs. ι climbs initially at the same rate for all tolerization periods, because there is no difference in the rates at which autoreactive detectors are eliminated. However, once new immature detectors have survived the tolerization period and become naive, ι starts to decrease, which is reflected by the drop-off in the various curves. As expected, the drop-off occurs at roughly the tolerization period, that is, T time-steps after the start of the simulation. Because of these dynamics, the tolerization period makes a significant difference to ι, for example, the final ι (after 12 weeks) is 0.01, averaged over 10 runs for $T = 1$ day, compared to 0.25 for $T = 28$ days. So, to reduce ι, short tolerization periods are to be preferred.

Both the false positive rates and the fraction of immature detectors can be reduced by using a non-zero match decay. In the results to follow, a match decay of $\gamma_{match} = 4 \times 10^{-5}$ was used, which is equivalent to a decay period of $t = 1$ day. A non-zero match decay reduces false positives because it reduces the time interval over which matches can be accumulated to reach the activation threshold; as the simulation period increases, this can make an increasing difference in performance. However, a match decay of 4×10^{-5} will have little effect on the detection rates of the real nonself test sets. The largest nonself test set is AP, which consists of a trace over 8600 time-steps; the probability of a single match decaying in this period is $1 - (1 - \gamma_{match})^8 600 = 0.29$, and for the next largest nonself test set, the probability of a single match decaying is $1 - (0.99996)^{3000} = 0.11$.

Using a non-zero match decay, there appears to be little difference between false positive rates for the different tolerization periods, as shown in figure 5.6. Once again, this figure shows typical runs for the five different tolerization periods, but now the false positive rates are closer together. However, the tolerization period still has a larger effect on ι. Figure 5.7 shows the change in ι over time for a typical run, for each of the five tolerization periods.

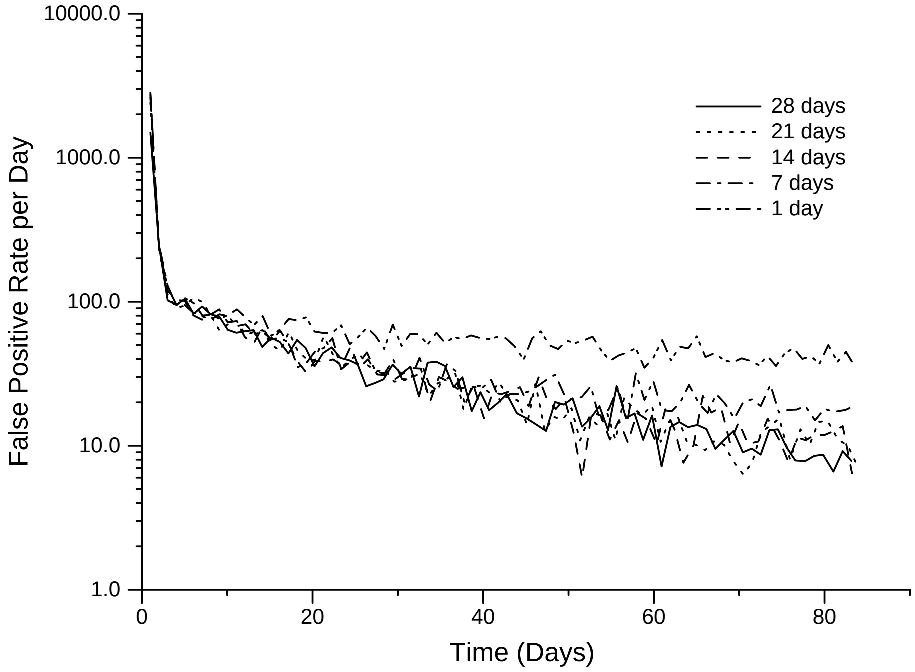

Figure 5.4: False positive rates over time for a typical run with different tolerization periods, for a massive self change (T varies). The legend indicates the tolerization period in days. Note that the y axis is logarithmic.

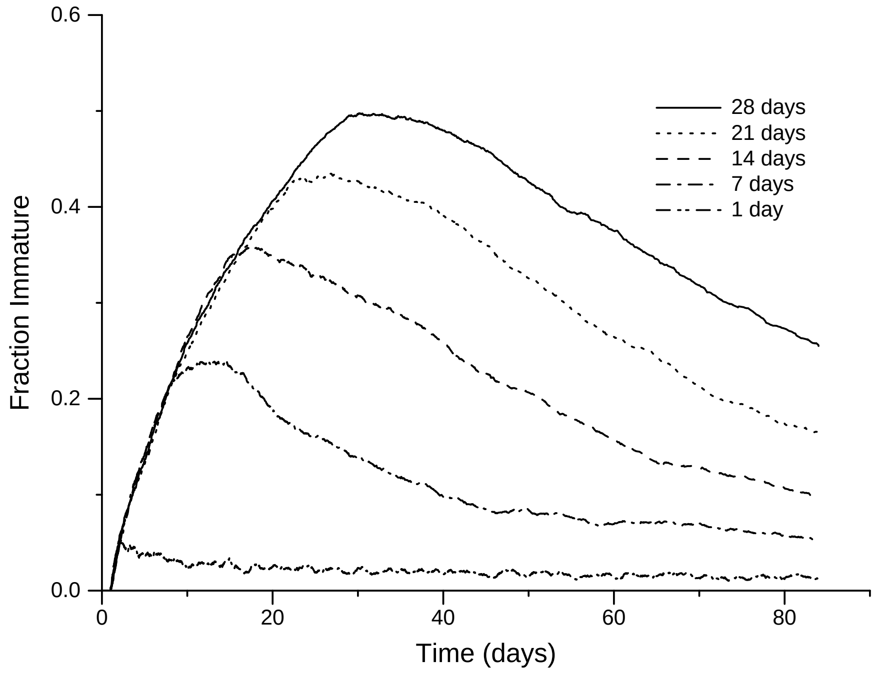

Figure 5.5: Fraction of immature detectors, ι, over time for a typical run with different tolerization periods, for a massive self change (T varies). The legend indicates the tolerization period in days.

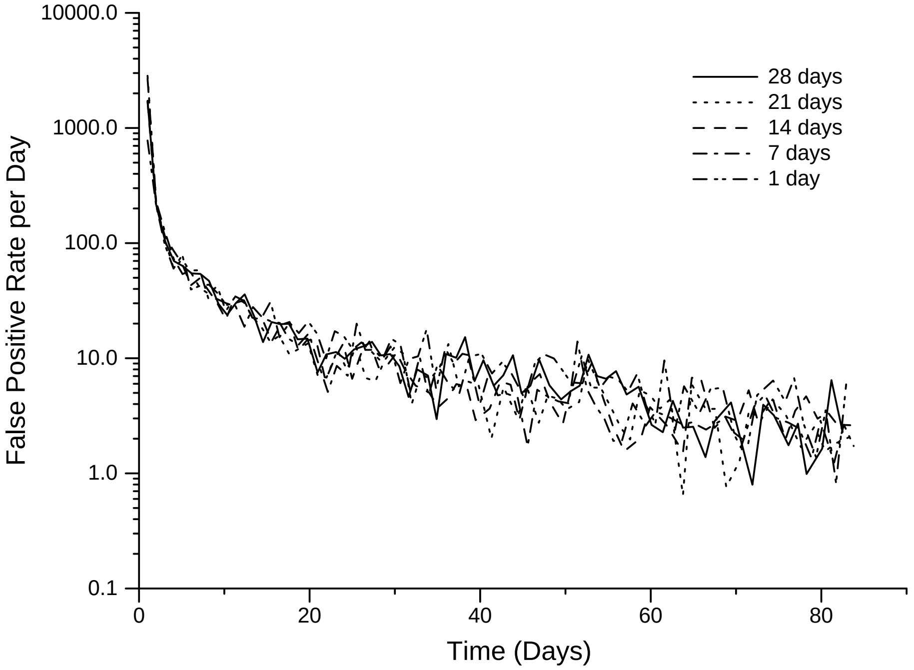

Figure 5.6: False positive rates, $\Delta\varepsilon^{+}$, over time for a typical run with different tolerization periods, for a massive self change (T varies, $\gamma_{match} = 4 \times 10^{-5} = 1/\mathrm{day}$). The legend indicates the tolerization period in days. Note that the y axis is logarithmic.

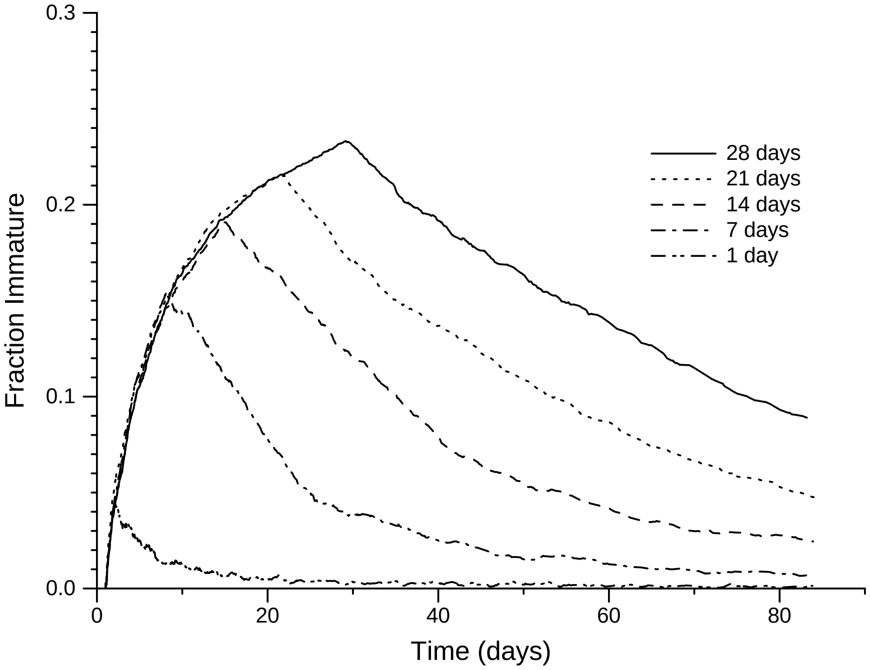

Figure 5.7: Fraction of immature detectors, ι, over time for a typical run with different tolerization periods, for a massive self change(T varies, $\gamma_{match} = 4 \times 10^{-5} = 1/\mathrm{day}$). The legend indicates the tolerization period in days.

T	$\Delta\varepsilon^+$ per day	Final ι	Max. ι	Max. Immature
1	3.8±0.4	0.001±0.000	0.048	2.1
7	2.4±0.2	0.007±0.001	0.152	8.1
14	2.4±0.3	0.022±0.001	0.195	15.1
21	2.4±0.2	0.047±0.002	0.222	22.0
28	2.1±0.2	0.082±0.003	0.241	29.0

Table 5.3: Effects of a massive change in the self set (T varies, $\gamma_{match} = 4 \times 10^{-5} = 1/\text{day}$). The simulation was run for 12 weeks, with the detectors initially tolerized off-line against S_{test}. The false positive rate, $\Delta\varepsilon^+$, is computed over the final two weeks of the simulation. The *Final* ι is ι at the end of the simulation, and the *Max.* ι is the maximum ι during the simulation, and the number of days after the start of the simulation at which this maximum was attained is given by *Max. Immature*.

Table 5.3 summarizes the different effects of the different tolerization periods, over 10 runs for each different value of T, when using a match decay period of $t = 1$ day. The final ι indicates that after a sufficiently long period of time (12 weeks), most detectors have matured, and in the case of the shorter tolerization periods (1 and 7 days), a negligible fraction of immature detectors remain (5 and 35 out of 5000, respectively). The maximum ι indicates the highest ι during the run, corresponding to the drop-off point on the temporal curve. For example, this drop-off happens when $\iota = 0.05$ if $T = 1$ day, but it reaches 0.241 if $T = 28$ days. The maximum immature point indicates the day in the simulation at which the maximum was reached and the drop-off begun. For all tolerization periods, this value is almost $T + 1$ days. The reason for the additional day is that the costimulation delay is one day, so any detector that is autoreactive will only get eliminated a day *after* the first time it is activated.

Note the difference between these results and those when $\gamma_{match} = 0$. The final false positive rate is less when using a non-zero decay, for example, even when $T = 1$, the false positive rate is half what it was when previously $T = 28$. ι is also reduced, because fewer detectors are matching, for example, the final ι for $T = 28$ days has been reduced to 0.082 from 0.25, and the peak ι is half what it previously was. So a non-zero match decay reduces false positives and the fraction of immature detectors.

In section 5.1.2 the accumulation of matches was modeled with queues. That analysis implied that as the tolerization period T grows larger than the decay period, t, the tolerization period should make a negligible difference, which is the case; when $T \geq 7$ (and $t = 1$), all false positive rates are about 2.4. However, when $T = 1$, the false positive rate is higher, at 3.8. The analysis further indicates that when T is close to t, the activation threshold τ should have little effect on the false positive rate. However, this does not take into account the fact that after every τ matches, the match count or queue size is reset to 0, so even for values of T close to t, it is expected that τ will influence the false positive rate. This is indeed the case, as is shown in table 5.4. When $T = 1$, $\Delta\varepsilon^+$ has increased from 3.8 for $\tau = 10$ to 6.6 for $\tau = 5$; when $T = 7$, $\Delta\varepsilon^+$ has increased from 2.4 to 3.8, so in this case the false positive rate has been *more* affected by a change of τ at shorter tolerization periods.

These experiments with a massive change in the self set are a simulation of what will happen if the distributed ID system is trained on a small sample of normal data (collected over a week), and then left to monitor all traffic for several months, without any further training. In other words, these mechanisms suggest a practical way in which an on-line system could be initialized.

T	$\Delta\varepsilon^+$ per day	Final ι	Max. ι	Max. Immature
1	6.6±1.5	0.002±0.000	0.079	2.0
7	3.8±1.0	0.014±0.004	0.233	8.0

Table 5.4: Effects of a massive change in the self set (T varies, $\gamma_{match} = 4 \times 10^{-5} = 1/\text{day}$, $\tau = 5$). The simulation was run for 12 weeks, with the detectors initially tolerized off-line against S_{test}. The false positive rate, $\Delta\varepsilon^+$, is computed over the final two weeks of the simulation. The *Final* ι is ι at the end of the simulation, and the *Max.* ι is the maximum ι during the simulation, and the number of days after the start of the simulation at which this maximum was attained is given by *Max. Immature*.

5.2.3 Memory

It was proposed that memory of past intrusions together with dynamic detectors could improve detection rates. This section demonstrates that this is indeed the case. Once again, ideas have been borrowed from immunology; a detector is the equivalent of a lymphocyte, and like a lymphocyte, a detector can become a long-lived memory detector, and has a finite lifespan.

The shorter the tolerization periods, the shorter the lifetimes of detectors, and the more useful dynamic detectors will be for providing changing coverage. The results in the previous section showed that a tolerization period of $T = 1$ day gave false positive rates of 3.8 as compared to 2.4 to 2.1 for longer periods, so these experiments were carried out using a tolerization period of $T = 1$ day, and a match decay of $\gamma_{match} = 4 \times 10^{-5}$, which is 1 per day.

The experiments performed here are simulations. First, the detection system is tolerized on S_{trn}, then it is run against a trace which is constructed as follows. At the start of the trace is the synthetic internal nonself test set, SI. Following this is a sequence of 250000 self strings generated by the random process X from the distribution of $S_{trn} \cup S_{test}$. Finally, the trace is ended with the same nonself test set, SI. The total time represented by the trace is approximately 10 days.

The goal of these experiments was to determine the effect of dynamic detectors and memory on detection rates when the system is re-exposed to the same nonself trace 10 days after the first exposure. Borrowing terms from immunology, the detection rate on the first exposure is called the *primary response*, and that on the second exposure is called the *secondary response*. Having self traffic in between tests the effect of dynamic detectors on false positive rates. The system was tested on SI because this is the hardest of the ten nonself sets to detect, and so memory and dynamic detectors should have the most influence on performance. The normal costimulation delay of 1 day was used, with the assumption that autoreactive detectors are never confirmed, and detectors that match nonself are always confirmed.

Table 5.5 shows the effects of different tolerization periods and different death rates on the detection system. The different parameters make a difference in the fraction of immature detectors, and in the false positive rates. ι is accurately predicted by equation 5.3, for example, when $T = 1$ and the expected lifetime, $E(\text{life}) = 7$, the prediction is 0.16 and the empirical value is 0.15. The predictions could be affected by elimination of autoreactive detectors (because these are not dying according to the death probability), but this number is so small (0.004; see table 5.2), that the effect is negligible. The secondary response reflects an improvement on the detection rates over the primary response which is typically over 3 times better; this indicates that dynamic detectors together with memory make a significant difference for detection rates. Although $m_d = 1$ for all experiments, the actual fraction of detectors that were memory detectors is lower, at about one in ten.

The false positive rates for the case of a tolerization period of one day ($T = 1$) are better (at $\Delta\varepsilon^+ = 3.5$ per day) than the values in table 5.2, where the false positive rate was 4.3 per day for a costimulation

T	E(life)	Memory Fraction	Primary Response	Secondary Response	Empirical ι	Predicted ι	$\Delta\varepsilon^+$ per Day
1	3.5	0.12±0.01	0.22±0.01	0.75±0.01	0.27±0.00	0.29	3.5±0.5
1	7.0	0.12±0.01	0.25±0.03	0.76±0.02	0.15±0.00	0.14	3.1±0.2
2	7.0	0.11±0.01	0.23±0.03	0.74±0.02	0.27±0.00	0.29	0.3±0.1

Table 5.5: Effects of tolerization periods and death probabilities on memory ($m_d = 1.0$, T varies, p_{death} varies, $\gamma_{match} = 4 \times 10^{-5} = 1/\text{day}$). *E(life)* refers to the expected lifetime, which is $1/p_{death}$. The *Memory Fraction* is the fraction of detectors that are memory detectors. The *Primary* and *Secondary* responses are the detection rates against SI, at the beginning of the simulation, and at the end of the simulation (after 10 days), respectively. The *Empirical* ι is the average fraction of immature detectors over the simulation, the *Predicted* ι is the fraction of immature detectors predicted by equation 5.3, and the $\Delta\varepsilon^+$ is the average number of false positives per day over the simulation. All values are averaged over 10 runs.

delay of one day and $\tau = 10$. This improvement can be attributed to using a match decay, because for the experiments reported in this section the decay period is one day, $t = 1$, whereas in the experiments reported in section 5.2.1 the decay period was infinite. Once a period of $T = 2$ is reached, the false positive rate is already so low that there is no point in increasing T further. However, there is a disadvantage to having longer tolerization periods: the lifetime must be increased to reduce the fraction of immature detectors. Longer lifetimes mean fewer new detectors over short time periods, which will make the system more vulnerable to repeated exploits of existing gaps in the coverage.

The effect of varying the tolerization period and keeping the decay period constant is shown in figure 5.8. The decay period was one day, and the expected lifetime of all detectors was seven days. This plot includes points where the tolerization period is less than the decay period; it is in this regime that the standard queuing theory fails, because it predicts that there will be no stable distribution for the length of the queue (number of matches). However, as figure 5.8 shows, there is a clear relation between the tolerization period and the false positive rate, given by

$$P(\Delta\varepsilon^+) = \alpha\beta^T \tag{5.4}$$

where α and β are constants, the false positive rate is per day, and the tolerization period is in days. The dashed line in figure 5.8 indicates the curve when $\alpha = 61.7$ and $\beta = 0.06$. A consequence of this relation is that the change in the tolerization period affects the false positive rate more slowly than it would in the case of simple queuing. With a simple queue, a change in tolerization period from $T = 1.25$ to $T = 1.5$ would result in a 6-fold decrease in false positive rate, whereas in the experiments the false positive rate is halved.

Table 5.6 shows the effects of limiting the number of allowable memory detectors. In this experiment, the expected lifetime was 7 days, and the tolerization period was 1 day. Three different memory limits were tested, $m_d = \{0.07, 0.27, 1.0\}$. Neither the fraction of immature detectors nor the false positive rates are affected by the memory limit. Memory could have an effect on the fraction immature because memory detectors live indefinitely; memory preserves detectors from dying and being replaced by new immature detectors. However, the fraction of detectors that are memory is low enough for this effect to be negligible. Furthermore, the memory limit is only reached for $m_d = 0.07$; for the other limits, the fraction of memory detectors never approaches those limits. In the case where the fraction of memory detectors is limited by $m_d = 0.07$, the secondary response is worse, dropping from 0.76 in the unlimited case, to 0.64.

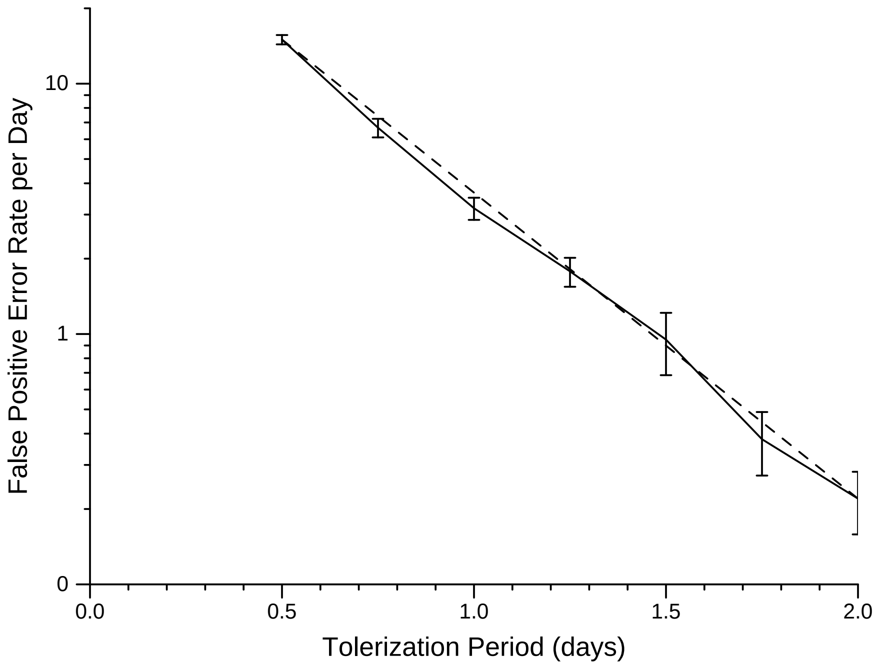

Figure 5.8: False positive rate per day with different tolerization periods (T varies, $\gamma_{match} = 4 \times 10^{-5} = 1/\text{day}$, $p_{death} = 5.7 \times 10^{-6}$). The solid line represents the experimental results, and the dashed line is the curve given by equation 5.4. Note that the y-axis is logarithmic.

m_d	Actual Memory	Primary Response	Secondary Response	ι	$\Delta\varepsilon^+$ per Day
0.07	0.07±0.00	0.24±0.02	0.64±0.01	0.16±0.00	3.4±0.3
0.27	0.10±0.01	0.22±0.04	0.73±0.02	0.16±0.00	3.3±0.3
1.00	0.12±0.01	0.25±0.03	0.76±0.02	0.15±0.00	3.1±0.2

Table 5.6: Effects of memory (m_d varies, $T = 25000 = 1\text{day}$, $p_{death} = 5.7 \times 10^{-6} = 1$ per 7 days, $\gamma_{match} = 4 \times 10^{-5} = 1/\text{day}$). The maximum allowable fraction of memory detectors is given by m_d and *Actual Memory* refers to the fraction of detectors which are actually memory detectors. The *Primary* and *Secondary* responses are the detection rates against SI, at the beginning of the simulation, and at the end of the simulation (after 10 days), respectively. ι is the average fraction of immature detectors over the simulation, and the $\Delta\varepsilon^+$ is the average number of false positives per day over the simulation. All values are averaged over 10 runs.

5.3 Summary

This chapter introduced four mechanisms (costimulation, distributed tolerization, dynamic detectors and memory), with three goals in mind: eliminating autoreactive detectors, adapting to changing self sets, and improving detection rates through signature-based detection. Costimulation is intended to eliminate autoreactive detectors, and was shown to reduce false positive rates from 8 per day to 3 or 4 per day. Distributed tolerization together with match decay allows the system to adjust to changes in the self set, with 3000 initial false positives per day dropping over time, down to 50 per day after a week, and 10 per day after 3 weeks. In the case where the self set is not undergoing a massive change, tolerization periods of 1 and 2 days, when used in conjunction with match decays, are sufficient to reduce the false positive rates further, for example, from 8 per day to one every three days. Dynamic detectors are useful for preventing repeated exploitation of existing gaps in detection coverage, and together with memory, improve detection rates for repeated attacks; detection rates improve from 0.25 for the primary response to 0.75 for the secondary, with one tenth of the detectors being memory detectors.

There is a trade-off between false positive rates and the fraction of immature detectors (which affects detection rates). This trade-off is adjusted by varying the tolerization period and the detector lifetime relative to the match decay period. Some aspects of this trade-off can be understood and some explanations for empirical data can be gained by modeling the match counts with simple queues. The conclusions to draw from this simple modeling are that the tolerization period should be greater than the decay period, but only moderately greater, because if the tolerization period is much greater, it will make little difference to the false positive rate. As the tolerization period is increased, so the lifetime must be increased to avoid having too many immature detectors (which will reduce detection rates). Longer lifetimes mean less dynamic coverage, and more opportunity for an attacker to exploit existing gaps in the detection coverage.

Chapter 6

Implications and Consequences

This chapter explores some implications and consequences of the results presented in previous chapters. Most of ideas discussed here could be viewed as future work; however, they do not necessarily follow on directly from the results already presented. Section 7.4 in the next chapter discusses future work that is directly related to previous results.

The first section (6.1) discusses automated response: ways in which the computer system could be designed to respond automatically to intrusions, with minimal input from a human operator. In section 6.2 two other domains are described in which the model of distributed detection would be useful. Finally, in the last two sections, some implications of the analogy are discussed, both for immunology (section 6.3.1) and for computer science (section 6.3.2).

6.1 Giving Humans a Holiday: Automated Response

The Immune System (IS) is an autonomous system: it detects pathogens and acts to eliminate those pathogens without outside interference or global control. The immune analogy can be extended so that the model of distributed detection described in chapter 3 includes automated response, and can be applied to make the ID system autonomous, so that the system takes care of itself, detecting intruders and taking steps to protect itself from those intruders, without the input of a human operator. Automated response can have negative effects, because any automated response that denies access to legitimate users is in effect a denial of service, making the system vulnerable to spoofing attacks in which an adversary stimulates the response system to shut off legitimate access. There is thus a trade-off between the harm done by the attacker and the harm done by the response system. The IS faces a similar problem, but unfortunately, what is known about response in the IS is too little to be of use as inspiration for artificial systems design (for an early model of this aspect, see [Segel, 1997]). Consequently, we must invent our own response mechanisms without the benefit of the biological inspiration.

6.1.1 Adaptive TCP Wrappers

A proposed basis for the automated response system is to automatically modify access to services by filtering out anomalous behaviour. This is a similar concept to having a dynamic or adaptive firewall, but each computer must be adjusted separately, and furthermore, to avoid single weak points, each computer should be able to filter its own traffic. This filtering can be achieved using TCP Wrappers [Venema, 1992], which are programs that "wrap around" individual services, and can be programmed to allow or disallow any connections or combination of connections to individual services. Each computer on the LAN would have TCP Wrappers for all its services, and automated response would take the form of modifying the wrappers to disallow traffic from suspicious sources.

The detection system would have to be modified so that each detector retains a list of all the strings that it has matched, that is, it retains information about which string caused each match in the match queue. Then a detector would have to retain information about at most τ strings. When the match count exceeds the threshold and the detector is activated, it sends out alarms to all internal computers that appear in any of the list of strings (at least one of the computers in each string has to be internal). For example, if a detector on computer A detects an anomaly that involves a connection from an external computer E to an internal computer B on the telnet port, then A sends a signal to B informing B that an anomalous connection was detected, coming from location E on the telnet service. A voting system could be used to ensure robustness: if B receives alarm signals from a sufficient number of other computers on the LAN about the *same* connection, then it assumes that the connection is indeed suspicious and modifies the relevant TCP wrapper to disallow those connections in future.

A problem with this automated response scheme is that false alarms can cause permanent denial of service to legitimate users, and as the experiments in this research have shown, new patterns are to be expected, although infrequently. So costimulation is still required to eliminate autoreactive detectors. Following the IS, costimulation should be provided in the presence of "damage" to the system, but for the moment, consider that costimulation is provided by a human operator, as described in section 5.1.1 (the issue of costimulation as a consequence of damage is discussed further on).

Each detector that signals an alarm posts a message to a human operator, called the System Security Officer (SSO) . The SSO inspects all the messages and decides which are true alarms. For those that are, the SSO sends messages to the computers that raised the alarms, confirming that they are indeed anomalous. If within the costimulation period, the detector on A that posted the alarm does not receive confirmation from the SSO, then the detector dies off and is replaced (as described in chapter 5). When a detector receives costimulation, it sends a message to B indicating that the classification was correct. Once again, voting is used: if B does *not* receive costimulation signals from a sufficient number of computers, then B reverts its TCP wrapper back to the original state.

There are 5 ways an attacker could subvert this system:

1. Ensure that the anomalous strings do not occur frequently enough to trigger an alarm.

2. Ensure that once an alarm is triggered, it is not confirmed by the SSO as an anomaly. There are two ways of doing this (short of physically incapacitating the SSO). Firstly, the attacker could somehow disrupt communication between the computers which detected the anomaly, and the SSO. Secondly, the attacker could try to launch the attack in such a way that the traffic is does not appear anomalous to an SSO, even if it is identified as anomalous by the detection system.

3. Compromise enough computers so that they can all send correction signals to the victim computer, causing it to modify its TCP wrappers in a manner favourable to the attacker. Note, however, that they could only cause it to modify wrappers to *correct* some mistake of the past, which means that the attacker must be interested in forcing the system to revert some block on past connections. The more computers required for a voting majority, the harder this would be to accomplish.

4. Compromise enough computers so that they all send alarm signals to the victim computer. Such attacks could only deny service to legitimate users, and once again, a voting majority of internal computers would have to be compromised. Furthermore, for the denial of service to be persistent, either the SSO would have to be subverted (because costimulation would be required for permanent effects), or the

alarms would have to be sent repeatedly, which will increase the chance that the SSO will become suspicious.

5. An attacker could completely compromise the system if communication to and from the SSO could be spoofed. Spoofing can be prevented by the use of public key cryptography to verify the parties in a communication.

A negative consequence of this scheme is that false alarms will lead to a denial of service, at least for the period of the costimulation delay. With alert security officers this could be minimised. The costimulation delay effects a trade-off between the debilitating consequences of false positives and the chance of missing an intrusion. This trade-off can be adjusted to suit the security policy of a particular LAN. One could imagine a system in which a user attempts to do something new, but legitimate, and triggers alarms. The user then waits (impatiently, almost certainly!), for the costimulation delay to expire, which may involve waiting, say, half a day, and then the user tries again, knowing that the system is now adjusted and will not signal alarms again. Of course, during this waiting, the user may well get a call from the SSO. A positive aspect of this scheme is that the system would react almost instantaneously to any intrusions, providing sufficient computers detected the anomaly. Messages could be dispatched and TCP wrappers modified rapidly enough so that suspicious connections are rejected before they can do harm.

Although this scheme automates response, it still requires a human operator (the SSO) to make the final decision as to whether behaviour is truly anomalous. Presumably the SSO make this decision by bringing into play additional knowledge about the usage of the system, the behaviours of typical computers and users, and typical methods and routes of intrusions. This suggests that the SSO could be replaced by some sort of expert system that uses knowledge about both the network, the software the computers are running, and typical intrusive activity to make decisions. This is an approach that has been taken in ID systems, but it is limited in that the automated response system must know something about what to expect; it is not as flexible as a human being who can make more broad-reaching inferences to reach informed conclusions. Unfortunately, for humans to do this successfully, they need to have expertise in computer security and knowledge of their own systems. Such experts are a limiting factor for all computer security systems that have to report to human operators. So how can we eliminate the human being from the loop, once and for all? Is this even possible?

In the immune system, one way in which costimulation is triggered is in the presence of damage. Cells in the body are continually dying and being replaced with new cells. In the course of normal functioning, cells usually die by apoptosis, or programmed cell death, whereas cells that die from damage (from pathogens, toxins, physical trauma), die a necrotic death. There is evidence to suggest that necrotic death produces different signals from apoptotic death. Such signals can be termed damage signals. Matzinger [Matzinger, 1998] proposes two classes of damage signals that are a consequence of necrotic death: *implicit* signals are given when proteins or other cellular material never normally seen outside the cell (even during apoptosis), are exposed to the immune system; and *explicit* signals are synthesized by cells that are under stress and likely to die a necrotic death, an example being heat-shock proteins which are given off by cells stressed by heat or cold, and some virus-infected cells, or those damaged by bacterial toxins. These explicit damage signals also give information about the nature of the threat that caused the harm.

How can these damage signals be mapped into a computer system? What is *damage* in a computer system? In the IS, damage is measured by a lack of functionality in the body; this translates directly into availability in a computer system: provided the system continues to function, the system has not been damaged. This definition is useful for identifying damage caused by Denial of Service (DOS) attacks, but it may not be useful when the damage is a consequence of loss of integrity or confidentiality. A possible explicit damage signal in such cases could be provided by host-based ID systems, such as Tripwire [Kim & Spafford, 1994] (which monitors critical files for changes), or an anomaly ID system that uses sequences of system calls such as that described in section 2.2. If the host-based ID system detects something suspicious, it provides the second signal required for costimulation. The problem with costimulation via explicit damage signals is that the victim computer, V, must itself provide the costimulation, and V may be compromised before it can provide the damage signal. If the damage signals can be subverted in this manner, then there is no way to determine whether V has been compromised, or "damaged". In this case subversion would have to take the form of preventing the damage signal from being emitted; any emission of a damage signal from V, even if it was the incorrect one, would alert the security system to the fact that V is potentially compromised.

What are the possibilities for implicit damage signals? The requirements are that these signals should indicate that the victim computer V is not functioning normally, and these signals should be hard to spoof if V is compromised. If availability is all we are concerned about, then the damage signal could be to test whether or not V is functioning at the normal level of performance. One way of doing this would be to communicate with V and time the responses. If V was sluggish in responding, or did not respond at all, then it would be assumed that V had been damaged, or compromised [Kaplan, 1998]. However, lack of availability may not make itself felt through the timing of network connections; what if all we cared about was the availability of processing power on V? Heavy usage of processing power need not affect the network communication speeds between V and other computers. In this case, the computer doing the checking, A, could have prior knowledge about the power available to V, and could query V to compute a function f, which A knows the answer to, and which A knows should take a given amount of time for V to compute. If f cannot be "guessed" or pre-calculated by V, then A can check on the processing power available to V, and hence determine if V is functioning normally. Because such an implicit signal is based on an actual functioning capability of V, it will be hard to spoof when V is not functioning normally.

Such implicit damage signals should work for detecting DOS attacks, but would not be useful for detecting compromises of confidentiality or integrity. What is needed is some indication of normal functioning, something that is equivalent to MHC in the body. MHC presents proteins from the interior of a cell to the immune system; similarly, a mechanism is needed to present indications of the internal state of a computer to the rest of the network. It is not clear what such a mechanism should be, and how it could be implemented to prevent subversion. Fundamentally, if an attacker in no way changes the internal state of a computer, then there can be no implicit damage signal. However, this is unlikely: an attacker usually has to modify some part of the system simply to gain access in the first place.

In conclusion, there are several implications of this proposed scheme of automated response. Firstly, each detector requires more storage (at most an increase by a factor of τ), to record past matches. Secondly, each local detection system would have to have sufficient detectors so that the detection rate per computer was high enough to form a voting majority. These first two points are feasible, considering how few detectors (100 per computer) are currently needed to globally detect all intrusions with clear separation, and considering that a voting majority need not be the majority of computers on the LAN, but only a sufficient number to make compromise of all of them extremely unlikely. The third implication is that damage signals are needed, either implicit, or explicit, or both. A few ideas for damage signals have been proposed, but in general this is an open question.

Finally, the implicit assumption behind this description of automated response is that the compromise of a single computer, or a few computers, is not critical; what is important is to prevent compromise of many computers. In other words, it has been assumed that individual computers are *disposable*. This assumption will not hold for all computer systems, particularly those concerned with the confidentiality of data on a single computer, unless sensitive data are stored in a distributed way such that confidentiality cannot be compromised without compromising many computers.

6.1.2 Fighting Worms with Worms

A worm is a program that automates intrusions so that it can spread itself from computer to computer. A worm must use the same attack routes as an human intruders, entering a computer system through vulnerabilities in services, guessed passwords, etc. The fundamental difference between a worm and a human intruder is that a worm is a *programmed threat* [Garfinkel & Spafford, 1996, chapter 11], and as such can spread faster than a single human could break into computers, provided there is some vulnerability that the worm can exploit. A virus is another form of programmed threat, but unlike a worm, a virus inserts itself into files on a system and is copied when those files are copied. Under the biological metaphor, a virus is aptly named, because it uses the copying machinery of the host to propagate. A worm, by contrast, has its own copying machinery, and so is more like a bacterium[1].

In [Garfinkel & Spafford, 1996], it is suggested that if you suspect that your LAN is under attack from a worm, you should sever all contact with the outside world until the worm is isolated and removed. There are two reasons behind this suggestion: the first is to prevent the worm from spreading to other networks, and the second is to prevent confidential information from moving outside your network. Nonetheless, the measure is drastic, and being disconnected from the outside world could certainly be regarded as a denial of service. It is proposed that a worm could be stopped if an automated response was used that detected the presence of the worm, and moved information fast enough across the network to prevent the spread of the worm. Not only are worms automated, but they also replicate, and so a replicating response is needed to combat worms.

One implementation of a replicating response would be to replicate detectors and move them around the network. To make the system effective requires that some characteristic signature of the worm be propagated across the network. Unless it is possible to characterise the signature in terms of a few targeted services, datapaths cannot be used as the signature, because the worm will continually be attacking from new locations and moving to new locations. It is more likely that the packets in which the worm is encapsulated will be characteristic in some way, so the solution would be to have a new class of detectors that match packet contents. These detectors will be mobile, moving from one computer to the next over the LAN.

Whenever a detector is on a computer, it will check all packets coming in to that computer. If a detector successfully detects a worm, it will make copies of itself, and those copies will be spread to other computers in the network. A form of affinity maturation can also be used to evolve detectors so they match the worm signature more closely. Affinity maturation may require an additional class of detectors, equivalent to Th-cells, that perform distributed tolerization. If the copies of an activated detector are memory detectors, then they will have the ability to detect and eliminate the worm without need for costimulation from a human operator. Hence they could spread across the network faster than the worm, because detectors spread by legitimate means, but worms have to find and exploit vulnerabilities before they can spread. If the spreading of detectors is sufficiently fast, the spreading of the worm can be halted by alerting computers beforehand to the presence of the worm[2].

Motile detectors would essentially be worms themselves, spreading by means of legitimate channels. If detectors become compromised or corrupted, they could become a security threat. Fortunately, the detectors will check each other, because any new detector that migrates to a computer will be checked by all detectors already on the destination computer. If corrupted detectors "look" different to normal detectors, then they will be detected and eliminated by other detectors. The IS functions in a similar way to eliminate corrupted lymphocytes.

[1] The term *bacteria* is also used to describe programs that replicate exponentially, without necessarily propagating across a network [Garfinkel & Spafford, 1996]. This is an unfortunate use of the term, which is more suitable for describing worms. Luckily, exponentially replicating programs are also called *rabbits*, so all is not lost.

[2] This is similar in spirit to the "kill" signal described in [Kephart, 1994].

In conclusion, there are several key questions that will have to be answered to determine if this idea is feasible. Firstly, do worms have characteristic signatures, in terms of packets contents? And, can a corrupted detector signature be distinguished from a normal detector signature? And finally, will these signatures be sufficiently compact that monitoring can be done in real-time? Real-time is critically important because worms can spread in real-time. Assuming the answers to these questions are yes, does this mean that other kinds of undesirable or intrusive behaviour can be detected (and stopped) in the same way?

6.2 Other Applications

The model of distributed detection presented and analyzed in chapter 3 is abstract and general, and so could be applied to many different domains. This section describes two other domains in which the model has the potential to be useful: mobile agent frameworks, and a certain form of distributed database.

6.2.1 Mobile Agents

Mobile agents can be defined as worms that have legitimate channels of access across networks. So a mobile agent is a piece of software or code that copies itself to networked computers, and then runs on those computers. The code can also be modified, or *evolve*, as a consequence of interactions with other agents and software on computers. Already there are several examples of commercial mobile agent frameworks; see [CACM, 1999] for examples of mobile agent uses.

There are no major applications using mobile agents, as yet. However, it is possible that they will become more important in the future because they offer increased flexibility for computing. For example, mobile agents could be used for performing expensive distributed computations, by buying "cycles" or processing power on under-utilized computers on other networks. However, flexibility comes at the price of increased vulnerability, both to malicious users exploiting the system, and to poorly programmed or accidentally corrupted agents that could spread and cause damage like a cancer.

Conventional models of computer security are not suitable for a mobile agent framework; application of such models could limit the flexibility to such a degree that the mobile-agent frameworks become mere curiosities. An example of this is the "sand-box" concept for Java applets. Java applets are mobile agents that run within a sand-box on a computer, where the sand-box is an interpreter that prevents the applets from harming the host computer. However, there are two problems with using a sand-box: firstly, the applets are limited in what they can do because they are constrained to the sand-box, and secondly, if what users care about is interactions between applets *within* the sand-box, then this form of security is useless. Essentially the sand-box is the same concept as the old fortress model of security, and although it is useful to a limited extent, it is generally too static to allow the kind of flexibility that makes mobile agents so promising.

It is suggested that mobile agent security can be implemented using a system similar to that described in the previous section for combating worms. A set of detector agents could be used, which would move around the network, from computer to computer, and monitor other agents to determine if any of them were anomalous. The detector agents could also have the ability to kill or otherwise incapacitate agents that were considered harmful. These detector agents are analogous to circulating IS cells, and other agents are analogous to cells in the body and pathogens.

It is not clear what should be used as the characteristic peptide for an agent. One possibility is something analogous to system calls. Every agent must use the equivalent of system calls when running on a host computer, for example, calls to the Java interpreter may be suitable for the peptide. Another possibility is to monitor the communications between agents. Presumably all agents will use a standard protocol to interact, and perhaps there are characteristics of messages in these protocols that will distinguish between acceptable and unacceptable behaviour. This may be a suitable peptide if the agents run in a sand-box, and it is the interaction between agents (which happens via communication using the protocol) that is of importance.

These proposed peptides are based on the notion of characterizing agents by their behaviour. An alternative is to arbitrarily assign a unique signature to each agent, and then learn to associate those signatures with behaviour, in other words, to use message digests such as MD5 hashes [Rivest, 1995] as peptides. This solution would be an easier option, because it is clear that agents can be distinguished on the basis of unique message digests. However, what is not clear is how these digests could be propagated when agents make copies of themselves, especially if the copies are modified or mutated. Having new digests is undesirable. Consider the case where the parent agent is a malicious worm, spreading across the network. As soon as the detection system has associated the characteristic digest of the parent agent with the harmful behaviour, all offspring of the worm can be traced and eliminated. However, if the digest of the worm changes completely each time it is copied, this is no longer possible. It is essential to pass on digests to offspring if memory detectors are to be useful.

An alternative idea is the notion of *digital epitopes* [Kaplan, 1998]. Each agent is signed with a unique digital signature, using public key cryptography [Diffie & Hellman, 1976]. The peptide would not be unique to a particular agent, but unique to a particular *originator* of agents (this could be a particular user or a particular computer, network, etc.). If an agent was identified as malicious then it is assumed that all agents signed by that originator are malicious. In other words, an originator is responsible for all its agents. This approach has the advantage that even if the code of an agent changes, the originator signature can remain the same. This approach is used by Microsoft for agent security; the digital signatures are called authenticode [Microsoft, 1999].

If there are imposed boundaries between agents, limiting interactions, then security will be easier, but interactions will be harder and the system less flexible. It is not exactly clear why we would want "maximum" interaction, but if we assume that this is the case, then we want minimal imposed boundaries, which means that we cannot stop agents from doing localized damage to other agents. However, using the distributed detection model, it will be possible to detect malicious agents and eliminate those responsible for damage, especially because agents operate in a localized sphere (a single computer). Damage signals could be implemented in this framework as follows. If an agent is terminated by another *non-detector* agent, then it could emit a signal indicating an unnatural death, whereas if it terminates itself, or is terminated legitimately by the sand-box controller or a detector, then it would not emit any signal.

The mobile agent framework is more suitable to the immunological model of distributed detection than network ID. Firstly, replication is an intrinsic part of the framework, and replicating detectors are essential to combat replicating agents. Secondly, clear damage signals can be derived. Thirdly, each "location" in the model is really a single agent, which means that each location has little memory compared to the set of self signatures, and so it becomes critical that detectors are generalizations of the self set, generalizations which have a low enough Kolmogorov complexity to be encapsulated within a mobile agent.

6.2.2 Distributed Databases

Another domain which is suitable to the model of distributed detection is one of a particular kind of distributed database. Consider a database that is distributed across many locations in a network. These locations can consist of computers connected in LANs or individual computers; these will be termed *systems* for the sake of simplicity. The network is not fully connected, so communication between systems at different locations may have to pass through multiple intermediate local systems, which will function as *routers*. Each part DB_i of the database can be modified only at a particular location, L_i, in other words, the only system with *authority* to modify DB_i is at location L_i. The system at L_i is called the *local authority* for DB_i. Each system on the network is a local authority for a different part of the database.

Any system at any location may require information from another local authority. For this it will have to query other local systems on the network. Ideally, it should query the local authority for the relevant part of the database, but to know who is the local authority for any arbitrary part of the database will require storing information about the mappings between all local authorities and parts of the database. If the number of locations is large enough, this will be impractical. However, it would be possible to have a single dedicated location, C, that has sufficient capacity to store all of this information. Then a local system, L_i, would only have to know the address of C so that it could query C for the desired information. This query from A would propagate through the network until it reached C, then C would know which local authority to query, so C could obtain the answer and send it back to L_i. If the query passed through the relevant local authority, L_j, then L_j could curtail the search and return the authorative answer.

Unfortunately, this scheme is not scalable and is not practical for large networks. C is a bottleneck: all queries have to go to C, and all replies have to be dispatched by C. This bottleneck is made worse by the fact that the locations are not fully connected, so multiple queries and replies will have to go through multiple locations, creating additional bottlenecks.

The bottleneck problem can be ameliorated through the use of local caches. Each local system has a cache which can store an amount of information that is small relative to the amount of information stored on C. Whenever a reply passes through a location, the local system will store the information in its cache. If the cache is full, it will delete the oldest information to make room for the new information. Now queries can be answered using information in local caches. As queries are chained back towards C, or towards a local authority, if any local system along the path has the answer, it can terminate the chain by sending the reply. This will relieve the bottleneck if sufficient queries can be answered from information in local caches.

This scheme works in practise. It is in use by the Domain Name System (DNS) [Mockapetris, 1987][3]. DNS is a distributed database that stores information about the mappings from IP addresses to domain names that are used throughout the internet. The database is distributed across the whole internet, and each local domain is the authority for its own mappings. The Network Information Centre (NIC) is the central location which stores information on which locations are local authorities for which part of the database. Queries which are made across the internet can be answered by local systems which have cached the relevant answers, or may have to chain all the way back to the NIC, which can then get the answers from the relevant local authority. Generally, DNS works well: answers to queries are usually prompt, considering the size of the internet.

The scheme described here is based on the assumption that local systems are honest, that is, they will not deliberately return corrupted information. In reality, some local systems may be malicious and corrupt data passing via that location across the network. The problem is exacerbated by the fact that corrupted information can spread. If a malicious system, M, returns corrupted information, then that information will be unwittingly stored in all local caches of honest systems along the path, which means that in future those local systems will also return corrupted information. In a well-connected network, corrupted information from a single malicious source could spread exponentially.

The problem of corrupted data can be solved using the immunological model of distributed detection. All replies to queries can be represented by message digests, such as MD5 hashes [Rivest, 1995]. These digests are compact so replies can be represented in a few bytes (typically from 16 to 64). The central location, C, retains a database of all valid digests for all local systems. This is presumably feasible because of the extreme compression of the digests. Every time a local system makes a change in the information within its authority, it notifies C. Public key cryptography [Diffie & Hellman, 1976] is used to verify the changes: each local system signs the changes with its private key, and C has public keys for all local authorities and so can verify that changes are legitimate. Of course, if the local information is changing frequently, this

[3] For more details on DNS, see [Albitz & Liu, 1992].

approach will not be practical. The assumption here is that changes to local information are less frequent than queries for information (this is the case with DNS).

The set of digests stored at C forms the set of self strings. C produces negative detectors which are tolerized against the set of digests. The negative detectors consist of randomly generated digests, together with a string matching rule to implement generalized detection. C continually dispatches detectors which spread randomly across the network from one location to the next. C signs all detectors using its private key, and every local system has a copy of C's public key, so every local system can verify detectors. When a detector reaches a location, it checks all the digests computed for information in the local cache, and whenever it matches, it informs the local system so that the corrupted information can be removed from the cache.

Detectors cannot be forged because they are signed by C, and each local system has C's key. A detector can only be autoreactive if it is outdated, that is, if information has subsequently changed and now the detector is deleting valid information. This kind of damage is different from the malicious spread of corrupted information, because all the detector is doing is removing information from local caches and so somewhat reducing the efficiency of queries and replies. The problem of autoreactivity can be reduced by giving detectors shorter lifetimes, so they are never too far out of date.

Replicating detectors can be used to stop the spread of replicating corrupted information, that is, replication can be used to "play the numbers game". When a detector is activated by matching corrupted data, it replicates itself and sends the copies to neighbouring locations. If the copies are exact replications of the original detector, then they can also be verified by C's public key and so will be recognized as legitimate detectors by local systems. The diffusion of the detector copies will follow network links, spreading down the same paths taken by corrupted information. The spread of detectors should catch up to the spread of corrupted information and halt it, because corrupted information only spreads in response to queries, whereas detectors can spread more rapidly under their own volition.

A problem here is that an autoreactive detector could also be activated and replicate, preventing the spread of legitimate information. Detectors may have short lifetimes, but the constant reactivation by legitimate information will cause replication and so could result in the indefinite retention of autoreactive detectors. This problem can be addressed by a form of costimulation. Each detector requires costimulation (or verification) from other detectors: whenever a detector encounters another detector, it checks to see if it matches that detector (which is possible if a symmetric match rule is used and detectors are represented by strings). If a detector does not match at least one other detector within a given period of time, it will die. This form of costimulation also allows for the implementation of memory detectors with indefinite lifetimes. Indefinite lifetimes are useful to detect when the same corrupted information starts to spread again, but the problem with indefinite lifetimes is that memory detectors could become autoreactive after some change in the database, and would need to be eliminated. This is where costimulation would be useful to eliminate even autoreactive memory detectors.

There are several key questions that would have to be addressed for this application. How many detectors would be needed to ensure that corrupted information could never spread enough to do much damage, and how much computational load would the detectors place on the system? Is costimulation really feasible? How many more detectors would be needed to ensure that costimulation works? Would it be possible and practical for C to store all digests, and for local systems to inform C of every change? Would it be better to have several locations at which detectors are produced, instead of one? Could some form of distributed tolerization be used so that there is no need for a single location, C?

6.3 Implications of the Analogy

This research leads to insights into aspects of both computer science and immunology. These insights are described here so of necessity this section will be vague, but it is appropriate at this stage to speculate. No concrete claims are made; rather this material is included to stimulate thought.

6.3.1 Understanding Immunology

The various mechanisms used in this research have specific effects and are essential for various aspects of the detection system. Do these mechanisms play a similar role in the IS, and are they equally indispensable? Consider the following:

- Central Tolerization (CT) generates detectors more rapidly than Distributed Tolerization (DT). DT works for tolerizing detectors, but it takes at least a day, if not several to tolerize the detectors, during which immature detectors are using resources and contributing nothing to detection. Furthermore, more rapid generation of detectors is particularly important when detectors are dynamic, because the more rapidly detectors can be generated, the more dynamic they can be. Is this equally important in the IS? If so, this would imply that the thymus is indeed an important organ, one which improves the efficiency of the system.

- If the self set is changing and/or cannot be sampled completely for the training set, then false positives will result because there will be autoreactive detectors. For this reason, frequency-based DT is essential. In the IS not all self peptides are expressed in the thymus, and the set of self peptides can change when the organism undergoes puberty and other life changes. This would imply that some form of frequency-based DT is necessary; this could be the costimulation of Th-cells by damage signals.

- In the network ID system, any form of autonomous response can cause denial of service if new, anomalous strings are not correctly classified . It was suggested that this problem could be solved using damage signals, so that anomalous strings are only classified as nonself if the detectors receive costimulation by damage signals. If the IS faces a similar problem of discriminating between infrequent, but valid self, and actual nonself, then damage signals could be crucial for eliminating autoreactive detectors. However, it is clear that having to rely on this feature alone would be limiting; the more detectors that can be effectively tolerized using CT without requiring damage signals, the better. The implication here for immunology, is that CT is necessary because of efficiency considerations, but frequency-based DT that is predicated on damage signals is also necessary to eliminate autoreactive detectors that cannot be tolerized in the thymus.

- Frequency-based DT is slow compared to CT, and will not work for B-cells adapting rapidly to pathogens during affinity maturation. The response must be as fast as possible; the longer it takes newly mutated detectors to mature, the more vulnerable the body. So it could be that the DT that is implemented by Th-cells providing signal II to B-cells is essential. Note that this mechanism of DT has not been used in the network ID system because there is no equivalent of affinity maturation.

- The closer the nonself test set is to the self set, the more multiple representations improve detection. In the domain of network intrusion it may not always be true that nonself is close to self, but in the body, pathogens will always be evolving so that they are more difficult to detect (they evolve towards becoming holes in the detection coverage). Those pathogens that are harder to detect will be the ones that survive better and hence get naturally selected. So it could be that in the IS, even more than in network ID, multiple representations are useful. How are these representations implemented in the IS

? One possibility is to view MHC as a representation. All MHC must bind to pathogens, but different types of MHC will bind to different pathogens. Therefore, the more more types of MHC there are, the better the protection. However, as the number of MHC types increases the number of immature lymphocytes eliminated in the thymus also increases, so there is a limit on the maximum number of MHC types. This trade-off is analogous to the trade-off between detection rates and retries that was used to evaluate matching rules and secondary representations. Increasing MHC types increases the detection rate, but increases the number of retries needed to generate detectors.

- If MHC types are implementing different representations, then the best MHC type would be one which spreads nonself and clumps self. Is there pressure on the IS to evolve these types of MHC ? How would one go about determining if this is actually the case ? In the experiments reported in this dissertation it was found that one randomly chosen representation was not much better than others, and that all contributed to the detection. Is this the case with MHC ? Perhaps so, if we imagine that MHC is evolving to clump self and spread nonself, but pathogens are evolving to be close to self, effectively cancelling the bias of the MHC.

6.3.2 Insights for Computer Science

This section explores some consequences implied by the effectiveness of multiple representations. Representation plays a role in multiple different domains, for example, search, planning, machine learning, etc. Representation is often essential and can influence the behaviour or performance of a system. Different representations exist for any given problem, but there is no theory that indicates in general whether we have chosen the correct or "best" representation. Indeed, there is no theory that indicates that there is such a thing as a "best" representation for any problem domain. The results of this dissertation suggest that choosing multiple representations (at random) and using them simultaneously is better than using any single representation. In some cases this could be problematic, for example, if we had to make a real-time decision between different options presented by different representations.

This discussion will be focused on the issues of representation in search; in particular, optimization over a cost function. Consider the problem of finding an algorithm for search that is general in the sense that it works better than other algorithms for a variety of functions, without needing prior knowledge of those functions. Such algorithms do not exist; it has been shown that the performance of any two algorithms is the same when averaged over all cost functions [Wolpert & Macready, 1995]. This result is known as the No Free Lunch (NFL) theorem. Effectively, it means that if algorithm A outperforms algorithm B on some class of functions, ϕ, then B will outperform A on the remaining functions, $\mathcal{F} - \phi$, where $\mathcal{F}$ is the (finite) set of all cost functions. So we cannot say that A performs better than B on average. The consequence of this is that one search algorithm can only be justified as better than another in the context of a given problem; we cannot consider the algorithms independently of the problem domain.

It is suggested here that using multiple representations simultaneously can overcome the limitations imposed by the NFL theorem. Any search algorithm requires some form of representation. This representation modifies or biases the cost function. So, searching simultaneously with multiple representations is equivalent to simultaneously searching multiple cost functions; the algorithm is applied to a subset of all cost functions, rather than a single function. On average, the NFL theorem still holds: we expect the performance of algorithms to be identical when averaged over all subsets of cost functions. However, if we are simultaneously searching through a subset of cost functions, then we can choose the one on which the algorithm performs the best (i.e. choose the best representation).

The performance of algorithms must now be evaluated not in terms of their average behaviour, but in terms of their *minimax* behaviour [Wolpert & Macready, 1995], which is defined by the distribution of algorithm performance over all cost functions. What we are interested in is the distribution of performance over the subset of cost functions that are formed by the multiple representations. If algorithm A has a distribution with strong variance (in other words, there are a few cost functions for which it performs well, and a few for which it performs badly), and algorithm B has a distribution with low variance (i.e. it performs more or less uniformly on all cost functions), then with multiple representations A will be better than B, because the performance of A is measured in terms of its best performance, which is better than B's best performance. It is hypothesised that when using multiple representations, there *are* better algorithms for general, no-prior knowledge search; these algorithms are ones which show the greatest variance in the distribution of performance on cost functions.

To illustrate this idea, consider two forms of search: a random walk, in which the search moves to a new point chosen randomly from the neighbourhood of the current point; and deterministic hill-climbing, in which the search moves to the best point in the neighbourhood of the current point. According to NFL, these two algorithms will perform equally when averaged over all cost functions. However, it is likely that random walk will perform similarly on all functions, that is, it will have a low performance variance over all cost functions. Deterministic hill-climbing, by contrast, is likely to perform well on some functions, and poorly on others; it has a high performance variance. When using multiple representations, if enough different random representations are used, there should be some representations for which hill-climbing will perform well for any problem. This is not the case for the random-walk; hence, hill-climbing is a better general, no-prior knowledge search algorithm than the random-walk.

Actually, high performance variance may not be sufficient. What is needed is that an algorithm exhibits high performance variance within a small sample of randomly-chosen cost functions, and in particular, the algorithm must exhibit instances of better than random performance within that sample of cost functions. Its not clear what the implications of this are for the distribution of performance over all cost functions. However, these ideas do fit in with the commonly-held intuitive notion that some algorithms are in general better than others, particularly if those algorithms are more successful than random on some problems.

Chapter 7

Conclusions

The goal of this dissertation was to describe and analyse an immunological model of distributed detection, and show how it could be applied to network ID. Towards this end a prototype network ID system was implemented, based on the immunological model. Several mechanisms of the IS were used, in an attempt to achieve the desirable immunological principles listed in chapter 1. The next section (7.1) revisits these principles and discusses how close the ID system has come to achieving them. The section following that (7.2) sums up the contributions of this dissertation, and section 7.3 discusses the limitations of this research. Several directions of future research are detailed in section 7.4, and the chapter is concluded with a final word in section 7.5.

7.1 Principles Attained

In section 1.3, nine principles were listed that guided this dissertation research. All of the principles have been attained, except *distribution* and *flexibility* in the strictest sense.

Distributed protection: No; a distributed system was not implemented, but rather simulated. Within the simulation this principle *was* achieved: detection was distributed across all simulated computers in the LAN, in the form of inexpensive (100 detectors per location) local detection systems, and almost all local detection systems contributed (roughly similar amounts) to global protection. This detection system has subsequently been implemented on-line, but not in a distributed manner.

Diversity: Yes; there are two forms of diversity: as a consequence of multiple random representations, and as a consequence of having different sets of randomly-generated, dynamic detectors on different computers. Both forms of diversity are useful: multiple representations ameliorate the problem of holes, and dynamic detectors ensure that gaps in protection change over time and so cannot be repeatedly exploited.

Robustness: Yes; loss or compromise of a single computer or a few computers will not cause a complete failure in detection, because all local detection systems contribute similarly to global protection. Furthermore, because there is communication (shared memory) is not necessary, compromise of a few computers will not cause an increase in false positive errors.

Adaptability: Yes, to a limited degree. The definition of adaptability used in section 1.3 was that the system should extract signatures from anomalies and refine those signatures to improve detection of the same anomalies in future. The system does extract signatures, but it only refines them to the extent that the memory detectors which implement signature-based detection have lower activation thresholds than normal. In another sense, different from the original one, the system is adaptable because it can adapt to changing self sets. This is an important point, one that was not made in section 1.3, because it is not clear that the IS does adapt to changing self. Generally, the body is static in that the set of expressed self peptides changes little over the life of the organism. This illustrates that we should not slavishly adhere to metaphors, but diverge from them whenever the particular application requires it.

Memory (signature-based detection): Yes; memory detectors implement signature-based detection, resulting in improved detection of previously encountered nonself strings.

Implicit policy specification: Yes; policy is defined through the normal behaviour of the network. Explicit policy is only incorporated to the extent that a human operator is required for costimulation. Mechanisms were proposed for removing dependence on the human operator all together, but in the absence of such mechanisms, the goal is to reduce dependence on the human operator, which the implementation of costimulation achieves.

Flexibility: No; in the definition of flexibility given in section 1.3, a system is defined as being flexible if it *automatically* adjusts resource usage to the current demands of its environment. The more computers are used, the more the system should curb its resource usage, and as they are used less, the system should increase its resource usage. This form of automatic adjustment was not implemented. Automated flexibility can be implemented for the prototype described here by varying the number of detectors, because both resource usage and detection rates are related to the number of detectors used. This is discussed in the future work section (7.4). Furthermore, the system is flexible in that performance is tunable in different ways.

Scalability: Yes; in so far as the system has no false positives. The system does have false positives, and it was shown that it is (and cannot be) scalable in terms of these errors. The definition of scalable used was applied directly to the concept of a distributed system: running the system across more computers does not increase computational costs or error rates. There are other definitions of scalable that this should not be confused with, in particular, whether a system is scalable with the problem size, which is reflected by the size of the self set. This all depends on if and how the number of computers increases with the size of the problem. For some problems the number of computers will be independent of the size of the self set; for the network ID problem, the size of the self set is directly related to the number of computers. In this sense, the system is still scalable provided false positives are minimized.

Anomaly detection: Yes; the ID system was able to detect all intrusions that were not included in the training set.

7.2 Contributions of this Dissertation

There are three primary contributions made by this dissertation.

1. Formalization and new analyses of the immunological model of distributed detection, originally developed in [Forrest, *et al.*, 1994]. There are several contributions here. Firstly, a framework for explicitly incorporating distribution was developed, and was used to to demonstrate that negative detection is both scalable and robust. Furthermore, it was shown that any scalable distributed detection system that requires communication (shared memory) is always less robust than a system that does not require communication (such as negative detection). Secondly, algorithms were developed for determining

whether a nonself instance is an undetectable *hole*, and for accurately predicting performance when the system is trained on non-random data sets. Finally, theory was derived for predicting false positives in the case when the training set does not include all of self.

2. Description and empirical testing of several extensions to the basic model of distributed detection. These extensions include: multiple representations to overcome holes; activation thresholds and sensitivity levels to reduce false positive rates; false-positive default costimulation with signal II provided by a human operator, for the purpose of eliminating autoreactive detectors; distributed tolerization to adapt to changing self sets and allow for the use of on-line, distributed detector generation without communication; dynamic detectors to avoid consistent gaps in detection coverage; and memory, to implement signature-based detection.

3. Application of the model of distributed detection to network ID. This includes implementations, analyses and experimental testing, demonstrating how useful the model is for network ID. Empirical results showed that the ID system detects all seven real intrusions tested, with false positive rates of one per three days, using no more than 100 detectors (binary strings) per computer. The system used both anomaly detection and signature-based detection; the latter improved performance over anomaly detection, when combined with dynamic detectors to nonself strings that had been encountered before.

7.3 Limitations of this Dissertation

There are several limitations to this work. Only limitations of the network ID application are discussed here.

Broadcast assumption: With switched Ethernet, switches are employed so that each computer only sees the packets destined for it. If each computer only sees the packets destined for it, then the distributed detection is useless (assumption 4 in section 3.1.3 is violated). There are alternatives; these are discussed in section 7.4.

Dependence on a human operator: This is a limitation although all current ID systems are limited in a similar way. In this dissertation an attempt has been made to reduce dependence on the human operator. Furthermore, ways of overcoming this limitation have been discussed in chapter 6, in the section on automated response (section 6.1).

Limited signature-based detection: The form of signature-based detection used in this ID system is limited in that signatures are only defined in terms of the origins of attacks, the targets of attacks and the service over which the attacks happen. If exactly the same form of attack (for example, exploiting a vulnerability in sendmail) comes from different locations, then this signature-based detection would only be useful if the memory detectors had generalized over a set of foreign addresses connecting over sendmail that included the different attacking locations.

Use of only TCP SYN packets: Intrusions can involve network traffic at different protocol layers, for example UDP. These would not be detected by a system that only monitors TCP traffic. Furthermore, some forms of attacks may be related to parts of the TCP stream other than the SYN packets, for example, stealth scanning that involves probing a network using FIN packets (these are packets which terminate a connection) because FIN packets usually do not show up in system logs. Stealth scanning with FIN packets would not be detected by this system.

Stealthy attacks: If activation thresholds greater than one are used to reduce false positives, then an attacker can evade detection in two ways. Firstly, by making anomalous connections occur infrequently enough so that they will never accumulate to the point where they could trigger a detector, or secondly, by ensuring that during any given attack the number of connections made is fewer than that required to activate a detector. As the frequency of connections decreases, so the skill and patience of the attacker must increase. This limitation does not apply if an organization has the resources to cope with the higher false positive rates that are a consequence of minimal activation thresholds.

Exploitation of normal paths: An attacker can evade detection if the intrusion follows only normal datapaths through the network. To do this the attacker must first gather information on those paths, and the only way to really do that is to break into a computer on the LAN. Secondly, the attacker must wait for some period of time to determine what legitimate paths are. Thirdly, only a few paths will be legitimate, restricting the opportunities for attack.

Varied server traffic: Traffic to and from servers such as WWW servers is excluded. Characterizing self in terms of datapaths will not work for services which are expected to connect to any possible other computer at any time. Hence any attack that exploits vulnerabilities in these services will go undetected.

Ambigious events: It was assumed that the problem domain can be divided into two sets of events: a set of legitimate and acceptable (according to some policy) events, termed self, and a set of illegitimate events, termed nonself. In reality, there can be events which are legitimate some times and illegitimate at other times. Such ambiguous events violate the assumption and cannot always be correctly classified by the ID system.

Assumptions underlying anomaly detection: The anomaly detection performed by the system was predicated on the notion that policy (under which the self and nonself sets are defined) can be implicitly inferred by observing the behaviour of the system, and assuming that either self occurs more frequently than nonself, or that there is some period of time during which the self set can be collected separately from nonself. This is not only a limitation of this research, but a limitation for anomaly detection systems in general: some assumptions must be made about the relative frequency of occurrence and/or distributions of the self and nonself sets.

There are multiple ways in which these limitations can be overcome. Some possible solutions are described in the next section.

7.4 Future Work

There are multiple directions for future work. Only those that are immediate extensions of the work presented in this dissertation are described here. This proposed future work is aimed at overcoming limitations of the original research, and enhancing understanding of the system through additional analysis. There are other directions for future work, including applications of the model to completely new problem domains. Some of these were discussed before, in chapter 6.

Monitoring other protocols: The network ID system should be extended to monitor other protocols, particularly UDP, because network attacks have been launched via UDP. Other protocols that could also be considered, such as the Network File System (NFS) . Another extension that could improve detection would be to extend the monitoring to include TCP FIN packets, so that stealth scanning could be detected.

Monitoring traffic frequencies and packet contents: One of the simplifications used was to ignore the contents of packets and the frequencies of packets traveling over a given connection. An avenue of investigation is to determine if a stable self set could be defined in terms of packets contents or packet frequency. Because the local detection systems are so lightweight and cheap (100 binary strings of length 49 bits per computer), there is room for increasing the scope of monitoring to look at other aspects of network traffic. Successful monitoring of these aspects of network traffic would be useful from several perspectives. It would enable the system to detect intrusions that do not perturb normal datapaths; in particular, it would enable the system to monitor services such as WWW servers and detect attacks launched against those services. Furthermore, it would allow for signature-based detection where the signatures are independent of location. This may be a more effective way of extracting signatures that generalize over intrusive behaviours.

Extending to non-broadcast networks: The research needs to be extended to address the issues raised by non-broadcast networks such as switched Ethernet. There are several possible ways of dealing with switched Ethernet. The first is to place the ID system at the gateways, routers and switching hubs. The main problem with this solution is that we lose the advantages of distribution, unless traffic destined for the same computer can pass through multiple gateways/routes/hubs. Even in this case, the system is not as robust, because there are fewer gateways/routers/hubs than there are computers. An alternative solution is to modify the routing and switching functions, so that all TCP SYN packets destined for a subgroup of computers are broadcast to all computers within that subgroup. This should have negligible impact on network performance, because SYN packets make up a small fraction of the traffic (less than one percent for the system studied here). Finally, if another form of "peptide" is used, such as packet contents or packet frequency, then different locations on a switched network may still see the same behavioural patterns, and no modification to the switched Ethernet would be needed. This is more like host-based ID.

Simple queues for matching: The effects of activation thresholds were analysed using queuing theory. Queuing theory was applied, with the assumption that the only way in which matches leave a queue is if they decay. This is not true in the network ID system: whenever the match count equals the activation threshold, the queue is reset to zero. Extensions are needed to cope with this variation on simple queues. A possible avenue of investigation would be to change the implementation so that the match count to did not go to zero, but behaviour was like a simple queue. The advantage of this is that no extension to simple queuing theory would be needed to analyse and predict the behaviour.

Similarity-based activation: A constant value was used for the match decay. Another possibility is to make the match decay period proportional to the strength of the match (in the contiguous bits rule, strength would be measured as the length of the contiguous bits, which would be larger than r). The effect of this would be that the more closely a detector matches a string, the more likely it is to be activated by that string, because it will retain those matches in the queue for longer. This is analogous to the IS, where the duration of binding between receptor and epitope depends on the strength of the bond. Using this mechanism, a form of adaptation can be implemented, where detectors are selected for the strength of their match to nonself strings. If these detectors are more sensitive to particular nonself without being any more sensitive to self then the distinction between self and nonself will be emphasized, making it harder for attackers to execute stealth attacks.

Implementation of automatic flexibility: Automatic flexibility requires that the ID system adjust itself so that it is always using the maximum resources available, without impairing the normal usage of the LAN. This can be done by varying the number of detectors in each local detector set according to the current system usage. Reducing the number of detectors is simple: randomly kill off the required number. Increasing the number of detectors is not so simple. Whenever new detectors are generated they will have to be tolerized. Such tolerization will take at least a day, which is pointless because the resource usage on a computer can change from one minute to the next. One possibility is to store on disk the maximum possible number of detectors that we may ever want to run, and load the detectors as desired. The detectors can be tolerized beforehand. As resources become more scarce (the computer becomes more heavily-used), detectors can be copied out of memory on to disk, and as resources become more abundant, detectors can be loaded from disk into memory. Effectively each computer has a virtual set of detectors, which can be paged in or out of memory as resources allow. There could also be an option to allow the detectors to forcefully consume more resources when the system is perceived to be under attack.

Analysing costimulation: It would be useful to have some theoretical understanding of costimulation.

Analysing the effects of sensitivity levels: No theory concerning the effects of sensitivity levels was developed in this dissertation. Furthermore, the experimentation with varied sensitivity levels was limited. More experimentation is needed and it should also be possible to model sensitivity levels as queues. The combined effects of activation thresholds and sensitivity levels could be analysed as the interaction between two queues.

7.5 A Final Word

A network ID system was presented that is successful at detecting certain kinds of intrusions, while maintaining false positive rates that are low by current standards (one every three days). The system could detect other kinds of intrusions, but it cannot be assumed that it could detect all intrusions. This system is not a panacea: it should be used as a compliment to, rather than a substitute for, other security measures such as access controls, cryptography, and host-based ID.

Although this system cannot not detect all intrusions, and attackers with sufficient guile can evade it, it is still useful. It makes it harder for attackers to abuse our computer networks, and it is flexible and lightweight enough not to inconvenience legitimate users. This is an arms race, and every advance we make is welcome; like the red queen in Alice in Wonderland [Carroll, 1871], we must keep running just to stand still.

This dissertation is about more than network ID. The model of distributed detection is applicable to several other domains; some of the possibilities were described. In addition, investigation of this model may yield new insights into both computer science and immunology itself. In a sense, the true goal of this research was to demonstrate how fecund the crossover between biology and computer science can be. This research has demonstrated that these cross-cultural excursions are indeed worthwhile, for the more perspectives we can gain on a problem, the better are our hopes of solving it.

References

[Albitz & Liu, 1992] Albitz, P. & Liu, C. (1992). *DNS and BIND*. O'Reilly and Associates: Sebastopol, CA.

[Anderson, *et al.*, 1995] Anderson, D., Frivold, T., & Valdes, A. (1995). *Next-generation Intrusion Detection Expert System (NIDES): A Summary*. Technical Report SRI–CSL–95–07, Computer Science Laboratory, SRI International.

[Anderson, 1980] Anderson, J. (1980). *Computer Security Threat Monitoring and Surveillance*. Technical report, James P. Anderson Company, Fort Washington, Pennsylvania.

[ASIM, 1996] ASIM (1996). Information security - computer attacks at deparment of defense pose increasing risks. GAO Executive Report - B266140.

[Balasubramaniyan, *et al.*, 1998] Balasubramaniyan, J. S., Garcia-Fernandez, J. O., Isacoff, D., Spafford, E., & Zamboni, D. (1998). *An Architeture for Intrusion Detection using Autonomous Agents*. Technical report, Department of Computer Science, Purdue University.

[Blakely, 1997] Blakely, B. (1997). The emperor's old armour. In *Proceedings New Security Paradigms Workshop*.

[CACM, 1999] CACM (1999). Special edition on agents.

[Carroll, 1871] Carroll, L. (1871). *Through the looking glass and what Alice found there*. Macmillan: London.

[Chaitin, 1990] Chaitin, G. (1990). *Information, Randomness, and Incompleteness, 2nd Edition*. World Scientific: Singapore.

[Chapman & Zwicky, 1995] Chapman, D. B. & Zwicky, E. D. (1995). *Building Internet Firewalls*. O'Reilly & Associates: Sebastopol, CA.

[Comer, 1995] Comer, D. E. (1995). *Internetworking with TCP/IP*. Prentice Hall: Englewood Cliffs, NJ.

[Crosbie & Spafford, 1994] Crosbie, M. & Spafford, G. (1994). *Defending a Computer System using Autonomous Agents*. Technical report, Department of Computer Sciences, Purdue University.

[Crosbie & Spafford, 1995a] Crosbie, M. & Spafford, G. (1995a). *Active Defense of a Computer System using Autonomous Agents*. Technical Report 95–008, Department of Computer Science, Purdue University.

[Crosbie & Spafford, 1995b] Crosbie, M. & Spafford, G. (1995b). Defending a computer system using autonomous agents. In *Proceedings of the 18th National Information Security Systems Conference*.

[Denning, 1987] Denning, D. E. (1987). An intrusion detection model. In *IEEE Transactions on Software Engineering* Los Alamos, CA: IEEE Computer Society Press.

[Denning, 1992] Denning, D. E. (1992). *Cryptography and Data Security*. Addison-Wesley Inc.

[D'haeseleer, 1995] D'haeseleer, P. (1995). *Further Efficient Algorithms for Generating Antibody Strings*. Technical Report CS95-6, Dept. of Computer Science, University of New Mexico, Farris Engineering Building, UNM, Albuquerque.

[D'haeseleer, 1996] D'haeseleer, P. (1996). An immunological approach to change detection: Theoretical results. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop* Los Alamitos, CA: IEEE Computer Society Press.

[D'haeseleer, *et al.*, 1996] D'haeseleer, P., Forrest, S., & Helman, P. (1996). An immunological approach to change detection: Algorithms, analysis and implications. In *Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy* Los Alamitos, CA: IEEE Computer Society Press.

[Diffie & Hellman, 1976] Diffie, W. & Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, 22.

[Forrest, *et al.*, 1996] Forrest, S., Hofmeyr, S. A., & Somayaji, A. (1996). A sense of self for UNIX processes. In *Proceedings of the 1996 IEEE Symposium on Research in Security and Privacy* Los Alamitos, CA: IEEE Computer Society Press.

[Forrest, *et al.*, 1997] Forrest, S., Hofmeyr, S. A., & Somayaji, A. (1997). Computer immunology. *Communications of the ACM*, 40(10), 88–96.

[Forrest, *et al.*, 1994] Forrest, S., Perelson, A. S., Allen, L., & Cherukuri, R. (1994). Self-nonself discrimination in a computer. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy* Los Alamos, CA: IEEE Computer Society Press.

[Frank, 1994] Frank, J. (1994). Artificial intelligence and intrusion detection: Current and future directions. In *Proceedings of the 17th National Computer Security Conference*.

[Garfinkel & Spafford, 1996] Garfinkel, S. & Spafford, G. (1996). *Practical Unix and Internet Security, 2nd Edition*. O'Reilly and Associates, Inc.

[Gray, 1992] Gray, D. (1992). The dynamics of immunological memory. *Semin. Immunology*, 4, 29–34.

[Grimmet & Stirzaker, 1992] Grimmet, G. R. & Stirzaker, D. R. (1992). *Probability and Random Processes*. Oxford University Press Inc.: New York, NY.

[Hamilton, *et al.*, 1990] Hamilton, W. D., Axelrod, R., & Tanese, R. (1990). Sexual reproduction as an adaptation to resist parasites. *Proceedings of the National Academy of Sciences of the USA*, 87, 3566–3573.

[Heberlein, *et al.*, 1991] Heberlein, L., Levitt, K., & Mukherjee, B. (1991). A method to detect intrusive activity in a networked environment. In *Proceedings of the 14th National Computer Security Conference* (pp. 362–371).

[Heberlein, *et al.*, 1990] Heberlein, L. T., Dias, G. V., Levitt, K. N., Mukherjee, B., Wood, J., & Wolber, D. (1990). A network security monitor. In *Proceedings of the IEEE Symposium on Security and Privacy*: IEEE Press.

[Heberlein, 1998] Heberlein, T. (1998). NID overview. http://ciiac.llnl.gov/cstc/nid/niddes.html.

[Helman & Forrest, 1994] Helman, P. & Forrest, S. (1994). *An Efficient Algorithm for Generating Random Antibody Strings*. Technical Report CS94-07, Dept. of Computer Science, University of New Mexico, Farris Engineering Building, UNM, Albuquerque.

[Hochberg, *et al.*, 1993] Hochberg, J., Jackson, K., Stallings, C., McClary, J. F., DuBois, D., & Ford, J. (1993). NADIR: An automated system for detecting network intrusion and misuse. *Computeres and Security*, 12(3), 235–248.

[Hofmeyr, 1998] Hofmeyr, S. A. (1998). *A Computer Scientist's Interpretation of the Immune System*. Technical report, Dept. of Computer Science, University of New Mexico, Farris Engineering Building, UNM, Albuquerque.

[Hofmeyr, *et al.*, 1998] Hofmeyr, S. A., Forrest, S., & Somayaji, A. (1998). Intrusion detection using sequences of system calls. *Journal of Computer Security*, 6, 151–180.

[Hunt, 1992] Hunt, C. (1992). *TCP/IP Network Administration*. O'Reilly and Associates: Sebastopol, CA.

[Inman, 1978] Inman, J. K. (1978). The antibody combining region: Speculations on the hypothesis of general multispecificity. *Theoretical Immunology*.

[Janeway & Travers, 1996] Janeway, C. A. & Travers, P. (1996). *Immunobiology: The Immune System in Health and Disease, 3rd Edition*. Current Biology Ltd.: London.

[Kaplan, 1998] Kaplan, T. (1998). Personal communication.

[Kephart, 1994] Kephart, J. O. (1994). A biologically inspired immune system for computers. In *Artificial Life IV*: MIT Press.

[Kim & Spafford, 1994] Kim, G. H. & Spafford, E. H. (1994). *Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection*. Technical Report CSD–TR–94–102, Dept. Computer Science, Purdue University.

[Kolmogorov, 1965] Kolmogorov, A. (1965). Three approaches to the quantitive definition of information. *Problems in Information Transmission*, 1, 3–11.

[Kumar & Spafford, 1994] Kumar, S. & Spafford, E. H. (1994). A pattern matching model for misuse intrusion detection. In *Proceedings of the National Computer Security Conference* (pp. 11–21). Baltimore, MD.

[Lehmer, 1949] Lehmer, D. H. (1949). Mathematical methods in large-scale computing units. In *Proceedings of the 2nd Symposium on Large-Scale Digital Calculating Machinery* (pp. 141–146). Cambridge, MA: Havarvd University Press.

[Lippman, 1998] Lippman, R. (1998). Lincoln laboratory xbmiintrusion detection evaluation. http://www.ll.mit.edu/IST/ideval/index.html.

[Lunt, 1993] Lunt, T. F. (1993). Detecting intruders in computer systems. In *Conference on Auditing and Computer Technology*.

[MacKay, 1993] MacKay, C. R. (1993). Immunological memory. *Advanced Immunology*, 53, 217–265.

[Marrack & Kappler, 1993] Marrack, P. & Kappler, J. W. (1993). How the immune system recognizes the body. *Scientific American*, 269(3), 48–54.

[Matzinger, 1994] Matzinger, P. (1994). Tolerance, danger and the extended family. *Annual Review in Immunology*, 12, 991–1045.

[Matzinger, 1998] Matzinger, P. (1998). An innate sense of danger. *Seminars in Immunology*, 10, 399–415.

[Meade, 1985] Meade, G. (1985). Department of defense trusted computer system evaluation criteria. National Computer Security Service Centre.

[Microsoft, 1999] Microsoft (1999). Authenticode white paper. http://msdn.microsoft.com/ workshop/ security/ authcode/ authwp.asp.

[Mitchison, 1993] Mitchison, A. (1993). Will we survive? *Scientific American*, 269(3), 102–108.

[Mockapetris, 1987] Mockapetris, P. (1987). RFC1034/1035.

[Moskophidis, *et al.*, 1993] Moskophidis, D., Lechner, F., Pircher, H., & Zinkernagel, R. M. (1993). Virus persistence in acutely infected immunocompetent mice by exhaustion of antiviral cytotoxic effector T-cells. *Nature*, 362, 758–761.

[Mukherjee, *et al.*, 1994] Mukherjee, B., Heberlein, L. T., & Levitt, K. N. (1994). Network intrusion detection. *IEEE Network*, (pp. 26–41).

[NetRanger, 1999] NetRanger (1999). Netranger web site. http://www.wheelgroup.com/netrangr/lnetrang.html.

[Oprea & Forrest, 1999] Oprea, M. & Forrest, S. (1999). How the immune system generates diversity: Pathogen space coverage with random and evolved antibody libraries. In *GECCO 99, Real-world Applications Track*.

[Osmond, 1993] Osmond, D. G. (1993). The turn-over of B-cell populations. *Immunology Today*, 14(1), 34–37.

[Paul, 1989] Paul, W. E. (1989). *Fundamental Immunology, 2nd Edition*. Raven Press Ltd.

[Percus, *et al.*, 1993] Percus, J. K., Percus, O. E., & Perelson, A. S. (1993). Predicting the size of the antibody-combining region from consideration of efficient self/nonself discrimination. In *Procedings of the National Academy of Science 90* (pp. 1691–1695).

[Piel, 1993] Piel, J. (1993). Life, death and the immune system, special issue. *Scientific American*, 269(3), 20–102.

[Porras & Neumann, 1997] Porras, P. & Neumann, P. G. (1997). EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings National Information Systems Security Conference*.

[Porras & Valdes, 1998] Porras, P. & Valdes, A. (1998). Live traffic analysis of TCP/IP gateways. In *Networks and Distributed Systems Security Symposium*.

[Potts, *et al.*, 1991] Potts, W. K., Manning, C. J., & Wakeland, E. K. (1991). Mating patterns in semi-natural populations of mice influenced by MHC genotype. *Nature*, 352, 619–621.

[Power, 1998] Power, R. (1998). 1998 CSI/FBI computer crime and security survey. *Computer Security Issues and Trends*, 4(1).

[Ptacek & Newsham, 1998] Ptacek & Newsham (1998). Insertion, evasion and denial of service: Eluding network intrusion detection. Secure Networks Inc., http;//www.secnet.com.

[Rivest, 1995] Rivest, R. (1995). RFC1321.

[Segel, 1997] Segel, L. A. (1997). The immune system as a prototype of autonomous decentralized systems. In *Proceedings of the IEEE Conference on Systems, Man and Cybernetics*.

[Smith, *et al.*, 1998] Smith, D., Forrest, S., & Perelson, A. S. (1998). Immunological memory is associative. In *Workshop Notes, Workshop 4: Immunity Based Systems, Intnl. Conf. on Multiagent Systems* (pp. 62–70).

[Snapp, *et al.*, 1991] Snapp, S., Brentano, J., dias, G., Goan, T., Heberlein, L., Ho, C., Levitt, K., Mukherjee, B., Smaha, S., Grance, T., Teal, D., & Mansur, D. (1991). DIDS (distributed intrusion detection system) — motivation, architecture, and an early prototype. In *Proceedings of the 14th National Computer Security Conference* (pp. 167–176).

[Somayaji, 1998] Somayaji, A. (1998). Personal communication.

[Somayaji, *et al.*, 1997] Somayaji, A., Hofmeyr, S. A., & Forrest, S. (1997). Principles of a computer immune system. In *Proceedings of the Second New Security Paradigms Workshop*.

[Staniford-Chen, *et al.*, 1996] Staniford-Chen, S., Cheung, S., Crawford, R., Dilger, M., Frank, J., Hoagland, J., Levitt, K., Wee, C., Yip, R., & Zerkle, D. (1996). GriIDS - a graph based intrusion detection system for large networks. In *Proceedings 19th National Information Systems Security Conference*.

[Steinman, 1993] Steinman, L. (1993). Autoimmune disease. *Scientific American*, 269(3), 75–83.

[Stevens, 1994] Stevens, R. W. (1994). *TCP/IP Illustrated*. Addison-Wesley: Reading, MA.

[Tonegawa, 1983] Tonegawa, S. (1983). Somatic generation of antibody diversity. *Nature*, 302, 575–581.

[Venema, 1992] Venema, W. (1992). TCP wrapper: Network monitoring, access control and booby traps. In *Proceedings of the 3rd UNIX Security Symposium*.

[Vigna & Kemmerer, 1998] Vigna, G. & Kemmerer, R. (1998). NetSTAT: A network-based intrusion detection approach. In *Proceedings of the 14th Annual Computer Security Applications Conference*.

[Warrender & Forrest, 1999] Warrender, C. & Forrest, S. (1999). Comparison of data modeling methods for sequences of system calls. In *IEEE Symposium on Research in Security and Privacy* Los Alamos, CA: IEEE Computer Society Press.

[Williams, 1991] Williams, R. H. (1991). *Electrical Engineering Probability*. West Publishing Company: St Paul, MN.

[Wolpert & Macready, 1995] Wolpert, D. H. & Macready, W. G. (1995). *No Free Lunch Theorems for Search*. Technical Report SFI-TR-95-02-010, Santa Fe Institute.