# An Artificial Immune System Architecture for Computer Security Applications

Paul K. Harmer, Paul D. Williams, Gregg H. Gunsch, and Gary B. Lamont

***Abstract*—With increased global interconnectivity, reliance on e-commerce, network services, and Internet communication, computer security has become a necessity. Organizations must protect their systems from intrusion and computer-virus attacks. Such protection must detect anomalous patterns by exploiting known signatures while monitoring normal computer programs and network usage for abnormalities. Current antivirus and network intrusion detection (ID) solutions can become overwhelmed by the burden of capturing and classifying new viral stains and intrusion patterns. To overcome this problem, a self-adaptive distributed agent-based defense immune system based on biological strategies is developed within a hierarchical layered architecture. A prototype interactive system is designed, implemented in Java, and tested. The results validate the use of a distributed-agent biological-system approach toward the computer-security problems of virus elimination and ID.**

***Index Terms*—Agents, artificial immune system, computer security, computer virus, intrusion detection.**

## I. Introduction

THE WORLD has become a more interconnected place. Electronic communication, e-commerce, network services, and the Internet have become vital components of business strategies, government operations, and private communications. Many organizations have become dependent on the wired world for their daily activities. This interconnectivity has also brought forth those who wish to exploit it. Computer security has, thus, become a necessity in the digital age.

While information dependence is increasing, the threat from malicious code, such as computer viruses, is also on the rise. The number of computer viruses has been increasing exponentially from their first appearance in 1986 to over 55 000 different strains identified today [3]. Viruses were once spread by sharing disks; now, global connectivity allows malicious code to spread farther and faster. Similarly, computer misuse through network intrusion is on the rise.

Current computer-security solutions are "reactive." They rely upon collecting and analyzing specimens of new viruses or intrusion signatures in order to update scanners with the means of detection. This approach results in a slow reaction time to new threats and is quickly becoming too much of a burden to update with the increasing number of new viruses and inventive network attacks that are discovered each day. In the past, computer-virus scan string updates were provided every two to three months; currently, vendors provide updates every few hours [4]. To overcome this problem, a self-adaptive computer defense immune system (CDIS) based on biological strategies is developed.

This paper presents the design of an artificial immune system (AIS) as applied to the computer-security domain.[1] The purpose of this paper is to describe research on developing a virus-oriented CDIS, supplemented with an initial investigation into the feasibility of adapting CDIS for network intrusion detection (ID) [8]. Background into computer-virus detection and network ID are given in Sections III and IV. Next, a string-matching function is chosen that provides the necessary coverage and specificity for these aspects of the computer-security problem (see Section VII). This matching rule is deployed within an AIS architecture. This architecture is defined and built based on an immune system model of operations defined in Section VIII. The resulting biological models are implemented though the use of distributed software agents. A completely modular approach is taken, which allows for the introduction of multiple detector agent types while leveraging the common infrastructure of the system for oversight, reporting, and repair (see Section IX). The agents are deployed within a hierarchical structure that provides system management features (see Section VIII-C). The detectors studied in this research are for file infector viruses and stateless packet-based network intrusions. The results from system testing are presented in Section XII, with particular investigation into the areas of efficiency and effectiveness.

## II. Artificial Immune Systems

There are several computational techniques that look to biology for inspiration. Some common examples include networks, evolutionary algorithms, and AISs or immunological computation [5]. The biological immune system (BIS) has been the target of considerable research interest in the medical community from which several theories of system behavior have been developed with the hope of improving human life. The use of immune system models to solve the computer-virus problem has been suggested by [5], [6], and [9]–[11]. Their application has also shown promise for ID [12]. Further ideas on utilizing a multilayer model of the immune system for ID was proposed in

Manuscript received December 28, 2000; revised May 21, 2001. This work was supported by the Defensive Information Warfare Branch of the Air Force Research Laboratory's Information Directorate (AFRL/IFGB). This paper is based on P. K. Harmer's and P. D. Williams' theses, submitted in partial fulfillment of the requirements for Master of Science degrees at the Air Force Institute of Technology, Wright-Patterson AFB, OH.

P. K. Harmer is with the Air Force Research Laboratory, Wright-Patterson Air Force Base, OH 45433 USA (e-mail: paul.harmer@wpafb.af.mil.

P. D. Williams is with the Air Intelligence Agency, Lackland Air Force Base, TX 78236 USA (e-mail: paul.williams@lackland.af.mil).

G. H. Gunsch and G. B. Lamont are with the Air Force Institute of Technology, Wright-Patterson Air Force Base, OH 45433 USA (e-mail: gregg.gunsch@afit.edu; gary.lamont@afit.edu).

Publisher Item Identifier S 1089-778X(02)06069-1.

[1]The paper assumes that the reader has a cursory understanding of biological immune system processes. Those wishing more information in this field are referred to [5]–[7].

[13] to provide defense-in-depth. Immunological computation has also been applied to other problem domains, not all of which are in the computer-security field. Some of the more interesting examples include anomaly detection in time series data [5], fault diagnosis [5], decision support systems [14], multioptimization problems [15], robust scheduling [16], and loan application fraud detection [17]. The similarity in all of these applications is that they utilize the pattern-matching and "learning" mechanisms of the immune system model to perform desired system features. A lot of theoretical groundwork in immunological computation has been completed, but only a handful of AISs have been built [12], [17], [18]. Additionally, none have implemented detectors from multiple problem domains in order to provide a defense-in-depth approach to computer security.

The BIS is made up of many different types of cells that are deployed in great numbers. These cells operate independently, yet in cooperation with each other through complex chemical communication mechanisms in order to protect the body from foreign invasion. This highly parallel and distributed structure of the BIS suggests that an integrated architecture can be viewed as a multiagent system (MAS), where separate functions are carried out by individual agents [14]. Furthermore, the general immune system features represent a model of adaptive processes at the local level, with useful behavior emerging at the global level [5]. This is similar to the description of MAS operations by the artificial intelligence community [19].

## III. Computer-Virus Detection

Computer-virus detection is the process of finding malicious programs residing on a computer system. This process is commonly referred to as antivirus (AV) even though more entities than viruses are often looked for as part of the search process. The term computer virus is often attached to unwanted code that does malicious activities on its host computer. Applying the term in this fashion is imprecise and misleading as viruses are actually only one form of "rogue code." Malicious code can take the form of a Trojan horse, virus, or worm.

A Trojan horse is a program the masquerades as one program, while it actually performs an entirely different task altogether [20]. Trojan horses are also programs planted and run unbeknownst to the user or administrator to provide a "backdoor" onto the system. A Trojan horse can be placed on a system through cooperation by unwary users, e.g., by opening executable attachments to email. A popular example is the Cult of the Dead Cow's "Back Orifice 2000," a stealthy persistent program that allows the perpetrator to remotely control many of the functions of a compromised Windows-based system [21].

A computer virus is a program that can "infect" other programs by modifying them to include a possibly evolved version of itself [22]. One distinguishing feature of viruses is that they are parasitic. They require a host to run them and to spread their viral code [23]. This is usually another executable program although other hosts, such as disk boot sectors, can be infected.

Computer viruses are usually classified by their method of infection. The common subclasses of viruses are file infector, boot sector, and macroviruses. The file infector is the type most commonly associated with the term computer virus. File infection viruses work by inserting their code into executable files, just as the biological virus works by inserting its DNA code into living cells [20]. The host file then executes the malicious code on behalf of the virus. Boot sector viruses attach themselves to specific areas of a disk that are loaded and executed on startup. By placing its viral code into the boot sector of the disk, a virus can gain control of the computer immediately upon bootup. This allows the virus to execute before anything can detect its existence [24]. The macrovirus is a section of code contained within an application document. The intent of this capability was to add automation capabilities to otherwise static documents. As a further boon to virus writers, macroviruses are much easier to write than before because macros use high-level languages and do not require specific operating system knowledge. Our current CDIS prototype is limited to file infector virus detection and elimination.

Worms are programs that execute independently with the distinguishing feature that they utilize a computer network in order to propagate themselves [22], [23]. They often take advantage of security or communications protocol loopholes in order to spread [20]. The first worms were built at the Xerox Palo Alto Research Center. They were designed to perform useful work in a distributed environment, such as finding idle resources [23]. These original worms would probably be called mobile agents today.

The Melissa virus is more accurately termed a worm as it used the features of Microsoft Exchange e-mail in order to spread itself across networks. More modern malicious code utilizes a variety of techniques, blurring the distinction among forms: Nimda uses e-mail as one of its several transport mechanisms, exhibits viral propagation on infected machines, and provides Trojan horse capabilities [25].

### A. Nature of the Problem

The most common method of identifying viruses are signature strings. A 16-B string has become the defacto AV industry standard. Researchers at IBM have shown that 16 B is sufficient to identify malicious code with a 0.5% false-positive rate (Type I error) [9]. These 16 B must also be crafted so that they find known viruses, thereby minimizing the false-negative rate (Type II error). The largest problem with creating a new computer-virus immune system (CVIS) is the generation of these signature strings or antibodies in an immune system. The problem is that only some of the $256^{16} = 3.4 \times 10^{38}$ combinations identify one or more valid viruses. Furthermore, if a valid string could be generated each microsecond, it would take a serial computer $1.08 \times 10^{25}$ years to generate them all. This methodology uses simple exhaustive search, but even the generation of strings through machine learning techniques has been shown to be highly combinatoric [26]. In general, the generation of strings and then testing their capabilities as virus identifiers is similar to the Boolean-satisfiability NP-complete (NPC) problem [27]. In this problem, a Boolean function is known (e.g., a function that describes one or more viruses) and the goal is to find the instantiation of function variables that returns a true value from the function. There may be one, many, or even no valid variable assignments that return true. The problem then

degenerates into enumerating all possibilities, which leads to its classification as NPC. This indicates that a polynomial-time algorithm does not exist for generating antibodies, so an approximation algorithm is the only choice.

Another issue with generating all possible scan strings is retaining them in memory or offline storage. Again, if 16-B strings are used, storing all of the signatures would take $(256^{16} \times 16\ \text{B} \times 1\ \text{MB})/(10^6\ \text{B}) = 5.4 \times 10^{33}$ MB. Even with the removal of known invalid combinations, this is far too large for current storage methods.

An AIS approaches these problems through the use of general detectors that cover a wider area of the search space; however, a parallel implementation of an antibody generation program is needed to reduce the high combinatoric burden and make the system practical. The large storage requirements can be reduced through creative methods, such as compression, but the sizes required also indicate the need for a distributed system. Finally, the increased connectivity of networked systems, which has aided virus spread in the past, can be used as a defensive weapon with a distributed CVIS. This not only provides a parallel framework for the antibody generators and a distributed file system for their storage, but also provides the capabilities to eliminate intruders as they enter the system. A collective antiviral self-defense organization is created by delivering improved inoculations across the entire system.

There are many different forms of malicious code attacking computer systems today. The variety and number of contemporary viruses makes complete detection difficult as shown by the combinatorics, although it is possible to design efficient systems by utilizing distributed and parallel computational environments. Many of the same problems and solutions can be found in the ID domain as well.

## IV. Intrusion Detection

In its purest form, ID is the process of identifying the presence of unauthorized access to an enterprise's computing resources. In practice, ID is broader and includes the detection of:

1) misuse/abuse—unauthorized activities by authorized users (e.g., accessing pornography, theft of information, using corporate resources for personal gain);
2) reconnaissance—determination of systems and services that may be exploitable;
3) penetration attempt—unauthorized activity to gain access to computing resources;
4) penetration—successful access to computing resources by unauthorized users;
5) trojanization—presence and activity of unauthorized processes;
6) denial of service—an attack that obstructs legitimate access to computing resources.

For ease of discourse, the terms "intrusion" or "attack" are used loosely to encompass any of the above conditions, except where further clarification is needed.

### A. Nature of the Problem

ID is a difficult problem for a variety of reasons. First, there is a large number of communication protocols in use [e.g., internet protocol (IP), internet control message protocol (ICMP), simple network management protocol (SNMP), transmission control protocol (TCP), user datagram protocol (UDP), hypertext transfer protocol (HTTP), and address resolution protocol (ARP)]. Each protocol is vulnerable to certain types of exploitation; some are similar among protocols, but many are unique. Second, there are many operating system, network service, and user-application vulnerabilities—intentional services, software bugs, and error-check omissions—that provide exploitation opportunities by unauthorized people or processes. These two factors together beget an enormous number of highly varied approaches for abusing computing resources.

The most common and straightforward approach to ID applies the basic virus detection model: pattern matching against a library of signatures. If a match is made, an alert is generated. Using a robust library can potentially produce a low false-negative rate; i.e., the ID system (IDS) would rarely fail to detect known intrusions (Type II errors). However, there are several problems with this approach. First, the amount of data and speed at which it moves can outpace the ability of an IDS to monitor all of the data. The result is that the IDS effectively takes random samples and can miss key information. Second, signatures are produced reactively; rarely are signatures created prior to an exploitation being observed in the wild. Third, there is a very high innocuous "noise" level on networks due to misconfigured services, user accidents, damaged/lost data packets, network management services, heartbeat information, and other activities unrelated to intrusion attempts. These contribute to a very high false-positive alert rate: detection of activities that match signatures, but are not part of an attack (Type I errors).

It is this high false-positive rate that makes ID based on the recognition of a singular event ineffective. Alerts can be generated by the IDS, but reacting to every alert consumes enormous time and resources, resulting in a self-inflicted denial of service. A physician usually does not make a diagnosis based on a single symptom and, like symptoms of a disease, multiple alerts need to be correlated and analyzed. Intelligent processing is required, not just to recognize the patterns of activity making up attack profiles, but, more importantly, to attempt to determine intent. Possibly the most difficult aspect of ID is that legitimate usage shares many of the symptoms of unauthorized activity. Normal activity patterns on the protected systems need to be considered during the analysis process.

The correlation of multiple alerts requires maintenance of state information. Each alert produced by a low-level detector (herein called a "sensor") contains information about the protocol or command used plus other parameters that provide context, such as source and target addresses. Since multiple attacks can occur simultaneously, related and unrelated alerts can be generated from multiple sensors and attacks can be distributed widely over time (among other factors), rigorous maintenance of state information can be an enormous task. Multiple competing hypotheses must be entertained in order to correlate alerts and deconflict activities, yet these hypotheses cannot be held indefinitely lest partially completed attacks consume the analyst's (human or automated) resources.

In summary, ID is tasked with discerning the occurrence of any of a large number of highly varied patterns of nefarious ac-

tivity within a massive amount of authorized normal and abnormal, but innocuous, activity.

It is unrealistic and self-defeating to attempt to solve all aspects of this problem at once, so we have concentrated on the proactive/predictive development of antibodies covering large portions of the network traffic space not populated with self data. Our prototype ID component generates signatures for deployment to network sensors and we intend that the responses of those sensors will be correlated by an analytical engine yet to be developed. The antibodies use 320 bits for a signature, comprising 29 of the possible data fields in a network protocol packet (see Section VI-B). These fields have a range of values from 1 to 32 bits. Limiting the protocols under consideration and ignoring for the moment the three large sequence numbers, the number of possible combinations is dominated by TCP traffic at $2.45 \times 10^{55}$. Clearly, this event space defies deterministic search, so stochastic search with hefty generalization is used to explore large sections of this space in the development of useful antibodies.

## V. Symbolic Problem Domain

The major objective of our prototype system is to detect the existence of nonself patterns within a potentially larger set of existing self patterns. The problem domain is over the set $X$ of finite-length symbol sequences. $X$ is typically represented as $X \in \{0,1\}^l$ or $X \in \{0 \ldots 255\}^{(l/8)}$, but the exact representation is an implementation detail. Set $X$ contains two subsets, self $S \subseteq X$ and nonself $N \subset X$ such that $S \cup N = X$ and $S \cap N = \emptyset$ [28]. For virus detection, the nonself patterns represent malicious viral code, while the self set is indicative of legitimate benign programs. In an IDS, nonself patterns represent IP packets from a computer network attack, while self patterns are normal sanctioned network service transactions and nonmalicious background clutter.

The task of the detection algorithm is the classification of an input pattern $I \in X$ as either self or nonself. Given an input string $I$: $I \in \{0,1\}^l$, a detector set $D$: $D = \{\alpha_1, \alpha_2, \ldots, \alpha_i\}$, where $\alpha \in \{0,1\}^k, \quad k \leq l, \quad i \in \aleph$, a matching function $f$: $f(I,\alpha) \rightarrow \{p : \Re \,|\, p \geq 0 \wedge p \leq 1\}$, and a matching threshold $\epsilon$, the classification as self or nonself can be made as

$$\text{match}(f,\epsilon,I,D) = \begin{cases} \text{malicious}, & f(I,\alpha) \geq 1-\epsilon \\ \text{benign}, & \text{otherwise.} \end{cases}$$

This detection methodology can generate two types of errors: Type I, or false-positive errors, and Type II, or false-negative errors. A false-positive error $\delta^+$ occurs when a member of the self set $S$ is incorrectly classified as malicious. Conversely, a false-negative error $\delta^-$ is the classification of a member of the nonself set $N$ as benign

$$(I \in S \cap \text{match}(f,\epsilon,I,D) = \text{malicious}) \rightarrow \delta^+$$
$$(I \in N \cap \text{match}(f,\epsilon,I,D) = \text{benign}) \rightarrow \delta^-$$

## VI. Antibody Generation

### A. File Infector Antibodies

The antibodies for detecting file infections are simple byte strings. These patterns are compared to the bytes within the computer file system. The signature bytes themselves are created by a pseudorandom number generator. Because of the algorithmic differences between the AIS and the current static string methods, it is unclear if the 16-B string length is appropriate for a CVIS. Therefore, our experiments include an examination of string length on the CVIS's effectiveness. Additionally, improvements to the random search method can be made, but they are not implemented currently. One technique would be to follow the biological model. The BIS generates antibodies by choosing random sections from five separate gene libraries [16]. In this way, the B cells are able to create more than 100 million unique antibodies [29]. A similar process could use computer-virus byte-fragment libraries combined in random ways to produce antibodies based on known nonself patterns. Variation in the antibody population or an adequate balance between known and randomly generation signatures is needed to avoid converging to only known signatures. This would in effect revert us back to the signature-based model and negate the value of an AIS.

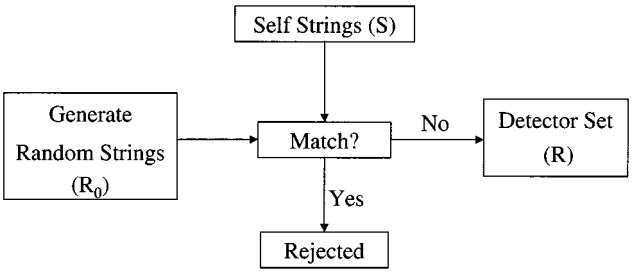


Fig. 1. Negative-selection algorithm.

Our pseudorandom antibody detection strings are assumed to be certified as nonself patterns via the negative-selection algorithm [30], [31]. This algorithm models the interaction and development of T cells in the BIS. Negative selection is used to precensor the generated antibodies against all known self patterns (see Fig. 1, [30]). This guarantees that false-positive errors do not occur against a static self as any antibody matching self is removed before fielding. Our system does not yet address the problem of a dynamic self space.

### B. Network Intrusion Antibodies

The antibodies for network intrusion are generated in the same manner and censored via negative selection just as in the AV detectors. However, the antibodies for network intrusion are longer and segregated because they utilize the IP packet structures as a template. There are many types of protocols flowing on our networks. For the purposes of this system, only the three most common protocols are monitored: TCP, UDP, and ICMP. All three of these protocols are layered on top of the IP. Network intrusion antibodies in the CDIS are essentially signatures for protocol packets. The fields in each of the protocols are mapped onto the first 292 bits of a 320-bit binary string (see Table I). The last 28 bits are used to determine whether a particular field is considered "valid" in the signature.

The generated antibodies, whether AV or ID, are deployed and compared to possible malicious attacks by a matching-rule function $f$. It is this function that provides the core functionality of the detection process. The proper selection of a pattern-matching function is instrumental in reducing the Type I and Type II errors.

TABLE I
NETWORK PACKET ANTIBODY MAPPING

| Gene | Field | Possible Values | Gene Bits | Start Loc | Valid Bit | Comment |
|---|---|---|---|---|---|---|
| **IP Fields (Common to all packets)** | | | | | | |
| 1 | Protocol type | TCP, UDP, ICMP | 2 | 0 | n/a | 1,2,3 correspond to TCP, UDP, ICMP respectively. 0 corresponds to dont-care |
| 2 | IP Identification number | 0-65535 | 16 | 2 | 292 | |
| 3 | IP Time to live (TTL) | 0-255 | 8 | 18 | 293 | Usually $\leq$ 128, small numbers are often interesting |
| 4 | IP Flags | 0-65535 | 16 | 26 | 294 | 2, 4 possible (and legal) all other possibilities suspect |
| 5 | IP Overall Packet length | 0-65535 | 16 | 42 | 295 | |
| 6 | IP Source Address A | 0-255 | 8 | 58 | 296 | |
| 7 | IP Source Address B | 0-255 | 8 | 66 | 297 | |
| 8 | IP Source Address C | 0-255 | 8 | 74 | 298 | |
| 9 | IP Source Address D | 0-255 | 8 | 82 | 299 | |
| 10 | IP Dest. Address A | 0-255 | 8 | 90 | 300 | |
| 11 | IP Dest. Address B | 0-255 | 8 | 98 | 301 | |
| 12 | IP Dest. Address C | 0-255 | 8 | 106 | 302 | |
| 13 | IP Dest. Address D | 0-255 | 8 | 114 | 303 | |
| **TCP Fields** | | | | | | |
| 14 | TCP Src port | 0-65535 | 16 | 122 | 304 | |
| 15 | TCP Dest port | 0-65535 | 16 | 138 | 305 | |
| 16 | TCP Sequence number | 0-4294967295 | 32 | 154 | 306 | |
| 17 | TCP Next Sequence number | 0-4294967295 | 32 | 186 | 307 | |
| 18 | TCP Ack number | 0-4294967295 | 32 | 218 | 308 | |
| 19 | TCP Flags | 0-255 | 8 | 250 | 309 | The sum of the flags in dec |
| 20 | TCPCWR | boolean | 1 | 258 | 310 | 1 = set, 0 = not set |
| 21 | TCPECN_Echo | boolean | 1 | 259 | 311 | |
| 22 | TCPUrgent | boolean | 1 | 260 | 312 | |
| 23 | TCPAck | boolean | 1 | 261 | 313 | |
| 24 | TCPPush | boolean | 1 | 262 | 314 | |
| 25 | TCPReset | boolean | 1 | 263 | 315 | |
| 26 | TCPSyn | boolean | 1 | 264 | 316 | |
| 27 | TCPFin | boolean | 1 | 265 | 317 | |
| 28 | TCP Packet size | 0-65535 | 16 | 266 | 318 | |
| 29 | TCP Data | 0-2047 | 11 | 281 | 319 | Not currently used |
| **UDP Fields** | | | | | | |
| 14 | UDPSrcPort | 0-65535 | 16 | 122 | 304 | |
| 15 | UDPDestPort | 0-65535 | 16 | 138 | 305 | |
| 16 | UDPLength | 0-65535 | 16 | 266 | 318 | |
| 17 | UDPData | 0-2047 | 11 | 281 | 319 | Not currently used |
| **ICMP Fields** | | | | | | |
| 14 | ICMPType | 0-255 | 8 | 122 | 304 | |
| 15 | ICMPCode | 0-255 | 8 | 130 | 305 | |
| 16 | ICMPDataLength | 0-65535 | 16 | 266 | 318 | |
| 17 | ICMPData | 0-2047 | 11 | 281 | 319 | Not currently used |
| **Gene Validity Fields (is a particular gene to be used?)** | | | | | | |
| | Gene 1 valid | boolean | 1 | 292 | | 1 = gene valid, 0 = not valid |
| | Gene 2 valid | boolean | 1 | 293 | | |
| | Gene 3 valid | boolean | 1 | 294 | | |
| | ... | ... | ... | ... | | |
| | Gene 28 valid | boolean | 1 | 319 | | |

## VII. PATTERN-MATCHING RULES

The BIS implements two core functions, the detection and elimination of pathogens, or harmful foreign invaders. This proposed CDIS is no different. The crux of the problem is the detection of malicious entities that have penetrated the boundaries of the system so that they cohabitate within a much larger self set. This is an application of pattern matching between a set of antibody scan strings and the set of input data.

Pattern recognition is a process by which input data are discriminated, not between individual patterns, but between populations. This is accomplished through a search for features common to members of the various populations or sets [32]. The BIS accomplishes this through the physical and chemical binding of antibodies to antigen molecules. Because of the negative-selection process, a match is a segregation of that molecule into the set of nonself. In the computational domain, this process is completed by string-matching rules; however, exact-equality Boolean matching does not produce the coverage and flexibility of the biological system. The model suggests using imperfect detectors to recognize nonself with a low false-positive rate and a high probability of detection.

### A. Matching Rules

The many pattern-matching functions come in two varieties: distance measures, which express how different two sequences are, and similarity functions, which measure how alike they are [33]. Intuitively, objects that are close together in the feature space must be similar, while those that are farther apart are dissimilar [33]. Those that are similar to a nonself pattern within a certain threshold can be classified as nonself. The matching rules investigated in this study utilize statistical, physical, and binary measures of distance or similarity. One statistically based similarity measure is the correlation factor or correlation coefficient.

*1) Statistical:* The correlation coefficient produces a number between $-1$ and 1 that relates how similar the two input sequences are. It is defined as

$$X, Y \in \{0 \ldots 255\}^N, \quad N = \frac{l}{8}$$
$$\rho = \frac{\sum_{i=1}^{N}(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{N}(X_i - \bar{X})^2 \sum_{i=1}^{N}(Y_i - \bar{Y})^2}}.$$

The most common implementation of this measure is $\rho^2$, which is somewhat easier to compute [33]. Other common matching rules operate at the bit level.

*2) Binary Distance:* The correlation coefficient utilizes the byte values of the input and antibody strings. However, at their lowest level these strings are sequences of bit values. Therefore, it makes sense to utilize difference and similarity measures that operate in the digital domain. The most obvious is the Hamming distance, which counts the number of bit features that are different between two strings. Taking the complement results in the number of bit positions that are alike [33]

$$\text{Hamming similarity} = \sum_{i=1}^{N} \overline{(X_i \oplus Y_i)}, \quad X, Y \in \{0,1\}^N.$$

The Hamming distance is the most commonly used method for measuring the distance between bit strings, but to be more useful, several authors have proposed additional similarity measures that extend the Hamming distance to produce the relative number of features that match or differ [33]. These matching functions utilize the following definitions:

$$X, Y \in \{0,1\}^N$$
$$a = \sum_{i=1}^{N} \zeta_i, \quad \zeta_i = \begin{cases} 1, & X_i = Y_i = 1 \\ 0, & \text{otherwise} \end{cases}$$
$$b = \sum_{i=1}^{N} \xi_i, \quad \xi_i = \begin{cases} 1, & X_i = 1, \quad Y_i = 0 \\ 0, & \text{otherwise} \end{cases}$$
$$c = \sum_{i=1}^{N} \gamma_i, \quad \gamma_i = \begin{cases} 1, & X_i = 0, \quad Y_i = 1 \\ 0, & \text{otherwise} \end{cases}$$
$$d = \sum_{i=1}^{N} \psi_i, \quad \psi_i = \begin{cases} 1, & X_i = Y_i = 0 \\ 0, & \text{otherwise} \end{cases}.$$

These basic measures are combined into many different similarity functions with the goal of producing a better similarity coefficient.

1) Russel and Rao
$$f = \frac{a}{a+b+c+d}.$$
2) Jaccard and Needham
$$f = \frac{a}{a+b+c}.$$
3) Kulzinski: A one has been added to the denominator of the author's equation to avoid division by zero errors. Due to the definition of $b$ and $c$, this occurs whenever there is an exact match
$$f = \frac{a}{b+c+1}.$$
4) Sokal and Michener
$$f = \frac{a+d}{a+b+c+d}.$$
5) Rogers and Tanimoto
$$f = \frac{a+d}{a+d+2(b+c)}.$$
6) Yule
$$f = \frac{ad-bc}{ad+bc}.$$

The final binary-distance function examined for system pattern matching is the $r$-contiguous-bits matching rule [34]. This rule attempts to model the strength of protein-antibody binding by equating longer substring matches with a higher affinity. Using this rule, string $X$ and string $Y$ are said to match if they agree in at least $r$-contiguous locations

$$X\text{: } ABA\underline{DCB}AB$$
$$Y\text{: } CAG\underline{DCB}BA.$$

In this example, $X$ and $Y$ match for $R \leq 3$ [30].

*3) Landscape-Affinity Matching:* The BIS "identifies" antigen by bonding with it physically and chemically. Only the correct inverse protein structure and chemical makeup binds with a high enough affinity to attach to an antibody or major histocompatibility complex molecule.

In most AISs, this binding is performed by bit or byte string comparisons [35]. Others extend bit matching to account for imperfect matches by using the Hamming distance or $r$-contiguous bits [12], [30]. Another extension is to present combinatoric variations of the nonself string to the detector in order to extend the search space of a specific matching function [12], [18]. All of these variations capture the chemical and physical matching process at a fairly high conceptual abstraction. Along with the historical matching rules, this study also introduces one more type, dubbed landscape-affinity matching.

In this methodology, the input strings are sampled as bytes and converted into positive integer values in order to generate a skyline, or landscape. The antibody strings are similarly represented. The antibody and input landscapes are compared in a sliding window fashion (see Fig. 2).

The comparison can be made in several ways that produce an affinity measure. Those used are difference, slope, and physical affinity. These measurements are then checked against a threshold value. If the affinity exceeds the threshold, a match

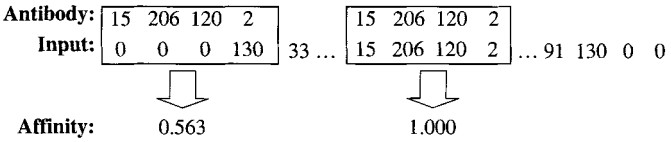


Fig. 2. Landscape-affinity-matching representation and windowing.

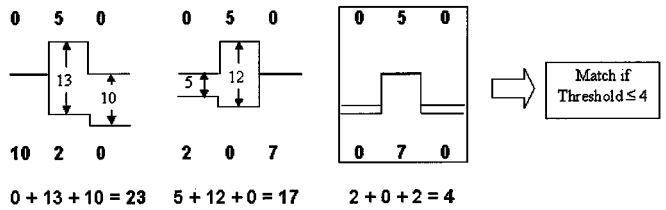


Fig. 3. Physical landscape-affinity-matching methodology.

is declared (see Fig. 3). The input string and the antibody are sequences of bytes compared $N$ at a time

$$X, Y \in \{0 \ldots 255\}^N.$$

In the difference-matching rule, the differences in the string bytes are simply summed

$$f_{\text{difference}} = \sum_{i=1}^{N} |(X_i - Y_i)|.$$

The slope-matching rule looks at the differences in the changes between bytes among the two stings

$$f_{\text{slope}} = \sum_{i=1}^{N-1} |(X_{i+1} - X_i) - (Y_{i+1} - Y_i)|.$$

Physical matching stacks the two strings like blocks and then calculates the resulting gaps between the two strings (see Fig. 3)

$$f_{\text{physical}} = \sum_{i=1}^{N} (X_i - Y_i) + 3 \times |\mu|$$
$$\mu = \min(\forall i, (X_i - Y_i)).$$

Landscape-affinity matching captures the ideas of matching the biochemical, physical structure, and imperfect matching with a threshold for activation. The differences between the input landscape and the antibody "heights" can be likened to the ease of chemical bonding between proteins. The closer the peaks and valleys are, the greater the likelihood of a bond and the higher the affinity.

### B. Comparison Criteria

A comprehensive theory on the probability of detection has been developed for the $r$-contiguous-bits matching rules in [34]. This theory was verified with experimentation in the AV domain in [30]. Our approach for this system is to compare the matching rules experimentally. This methodology was selected due to the relatively large number of rules and our desire to select a matching rule for a computer-security system, as opposed to developing a complete theory behind each of the rules. In order to compare these 12 selected matching rules, each one is calculated with a common data set. A random string of 32 B is generated as the input string. From this, 4 B are selected from positions 11–14 to act as an antibody string. Therefore, a known exact match is always present at position 14. The 4-B antibody

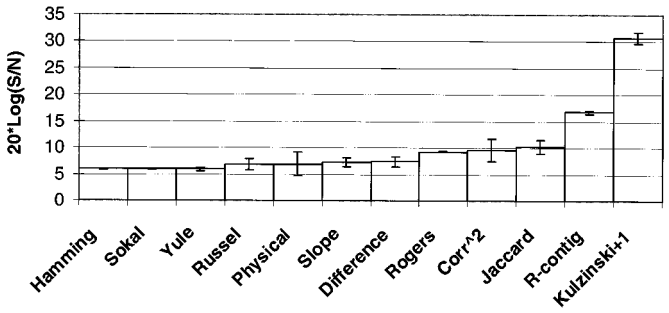


Fig. 4. Average SNRs.

is compared with the zero-padded input string using a sliding window. This generates 35 measurements of difference or similarity for each matching rule.

All measurements are converted to similarity measurements and normalized so that a value of one represents an exact match, while a zero is produced by the two most dissimilar strings. This test is run on five random input strings to produce a statistical sampling of the rules' performance. In order to compare the effectiveness of the various methods, an average signal-to-noise ratio (SNR) is calculated, along with a function-value distribution.

### C. Results and Analysis

The SNR is a measure of a matching rule's ability to accurately discriminate a match signal from all the nonmatches (noise). It is calculated as ten times the log of the ratio of the signal power to the average noise power. In order to equate with communications theory, in this application, the normalized rule function values are interpreted as voltages driving a normalized resistor. The result is a continuum of values from zero to one. A detection or a Boolean "match" value is an independent calculation and determined during scanning if the signal strength is greater than the predefined threshold value

$$\frac{S}{N} = 20 \times \log\left(\frac{1}{\eta}\right), \quad \eta = \frac{1}{N-1} \sum_{i=1}^{N-1} x_i, \quad x_i \neq 1.$$

The results can be seen in Fig. 4. A large SNR indicates a more specific detector, while a low value is indicative of a general detector. A specific detector is able to find a pathogen with a low false-alarm rate. As the SNR decreases, the probability of generating a false-positive detection increases. However, a general detector is able to cover a larger subset of the self/nonself space. This must be balanced with an appropriate affinity threshold value. Together, the matching rule and the threshold define the specificity of the detection process. The inherent tradeoff is between accuracy and coverage.

The Kulzinski measure produces a disproportionately large SNR. This measure would produce the most specific detector. The Hamming distance and the Sokal functions produce the lowest SNR, pulling a signal only about 6 dB above the noise floor. These would result in much higher false-alarm rates on average. Interestingly, the landscape-affinity measures did not perform much better. The $r$-contiguous-bits rule produced a SNR of almost 17. The increased stringency in this rule compared to the Hamming distance, where matching can occur anywhere, results in a detection rule that is almost three times more specific than the Hamming distance.

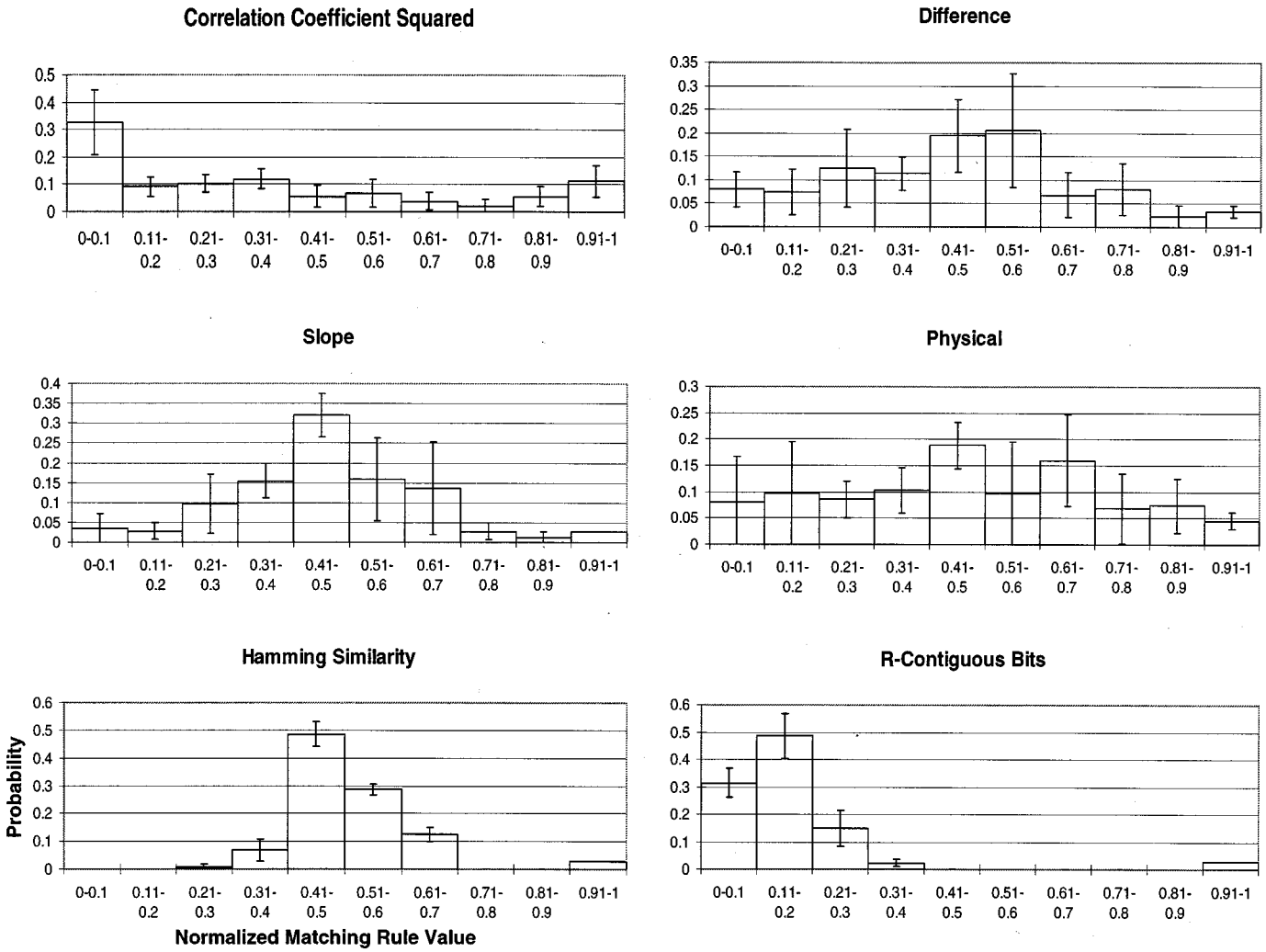


Fig. 5. Normalized-matching rule distribution functions, part I.

For this application, a balance between generality and specificity in the detector is desired, with a tendency toward the specific. A general detector allows the antibody to cover a greater portion of the nonself region, at the expense of possibly overlapping a small portion of self. Since we are using the negative-selection approach, the impact of general detectors would be a greater difficulty in generating the required number of antibodies (due to a higher probability of a match on self). Additionally, small areas of nonself (holes) in the landscape could be overlooked [36]. Another design goal for the CDIS is to increase the sensitivity of the detector by reducing the detection threshold. This allows the system to increase its awareness for a possible infection based upon outside notification or the recent occurrence of an attack. For this reason, a matching rule with the ability to pull the signal out of the noise floor, but not too high, is desirable. A SNR between nine and 12 is probably sufficient, which corresponds to the Rogers correlation-coefficient-squared and the Jaccard measures. In order to down select among these, the function value distributions are plotted.

The various values produced by the comparison functions are scaled and plotted using histograms in order to understand the density functions of the various measures. These can be seen in Figs. 5 and 6. Ideally, the density function for this application should approximate Fig. 7. This corresponds to a SNR of 8.05 dB. The ideal density function would allow for a low false-positive rate with a smooth scaling in sensitivity as the detection threshold is moved to the left. In the ideal case, the density function value at 90%–100% should be $(1/35) = 0.0286$, which indicates only one exact match and all other similarity values are less than 90%. Additionally, a low variability, especially in the higher affinity values, is desired. This would indicate consistent performance from the detector.

Evident in these histograms are the reasons for some of the SNR values as well as confirmation of the generality or specificity of the matching rules. The Kulzinski measure's histogram dramatically depicts this rule's ability to perform as a highly-specific detector. Likewise, the $r$-contiguous-bits rule is quite heavily weighted at the lower affinity end. The landscape-affinity physical measure produces the worst discriminator, with a naturally high false-positive frequency along with large variability in all the other value bands. Additionally, it has an almost uniform distribution in value frequencies, indicating poor discrimination. The Hamming distance and Sokal's measure, which possess identical SNR values, also show their equality in their density functions. Further investigation reveals

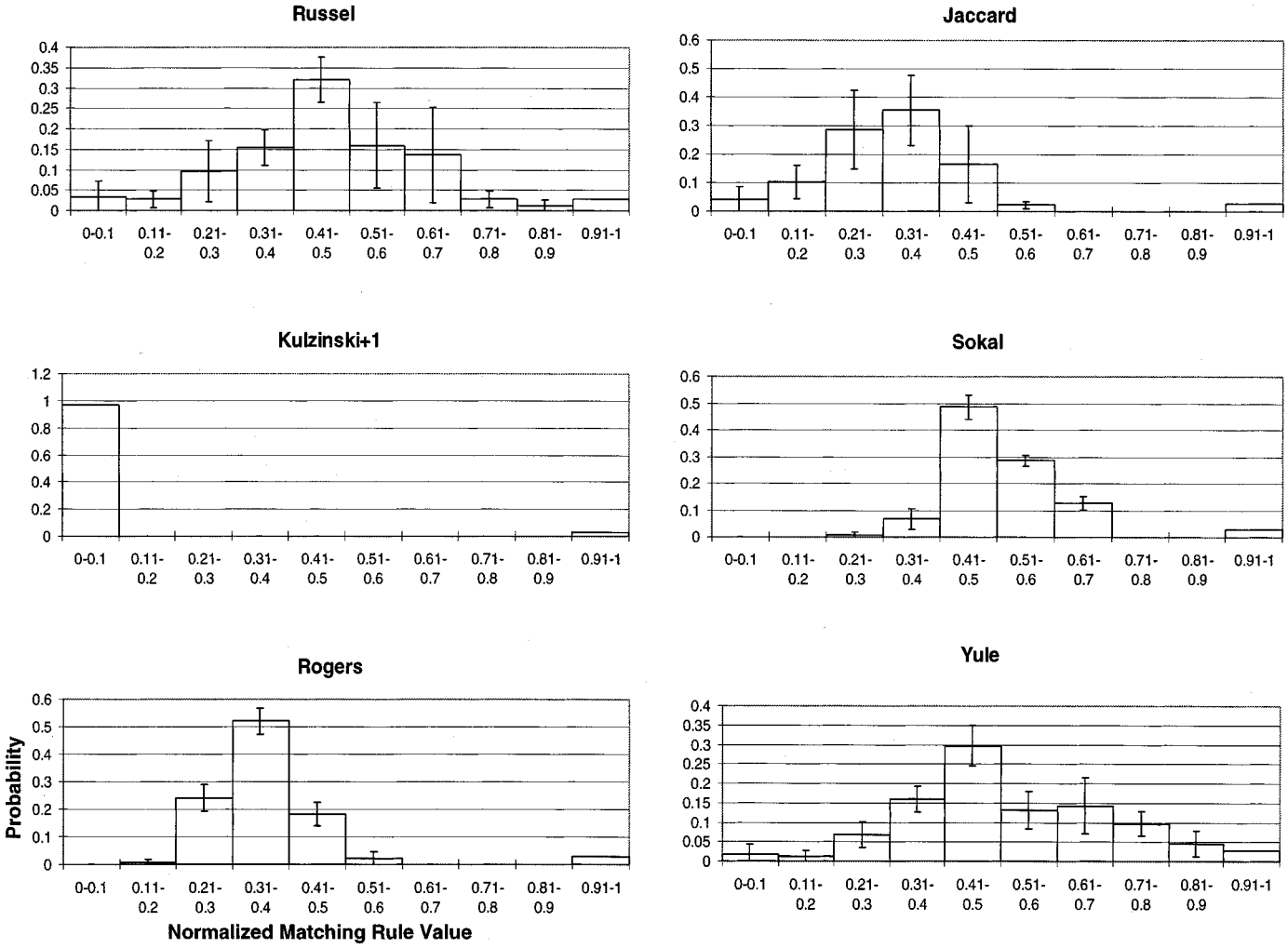


Fig. 6. Normalized-matching rule distribution functions, part II.

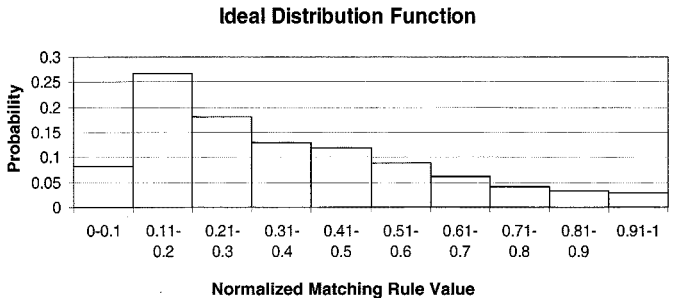


Fig. 7. Ideal-matching rule distribution function.

that the Sokal and Michener function is equivalent to the normalized Hamming similarity.

Based on their SNRs, Rogers, the correlation coefficient, and the Jaccard measurements are the most applicable to this application. Both the Rogers and Jaccard rules produce distribution functions that are only slightly better than the Hamming distance in terms of specificity. These two also have large gaps in their frequency distributions between an exact match of one and the values of lesser matching affinity. The Rogers measure is the best of the two because of its low variance in frequency values. The correlation-coefficient-squared produces a close-to-uniform distribution, which would scale well in sensitivity, but it has a very high false-positive frequency. This false-alarm rate renders the correlation coefficient unacceptable. For these reasons, the Rogers and Tanimoto measure is the best choice. Its density function is a fairly good approximation of the ideal case, but its greatest deficiency is the gap between a positive match and the next lowest frequency band. This either needs to be accounted for with a scaling of the threshold reduction or it allows for a sensitivity gap where the threshold would have to be reduced 40% before additional sensitivity is encountered. Heightened sensitivity could also be gained by the replacement of the Rogers function with the Sokal function. This would give the system the same performance as the Hamming distance if more generality is required.

The Rogers and Tanimoto similarity measure is the best matching rule for this application. It provides a good compromise between a specific versus a general detector and can also accommodate increased sensitivity through detector threshold reduction, although a fairly large reduction is required.

For each of these matching rules, additional mathematical operations such as squaring, scaling, or taking the absolute value can have a dramatic effect on the density function histogram and the SNR. The Yule discriminator produces a value between $-1$ and 1. Scaling with the absolute value produces a density function that almost exactly matches the ideal case. However,

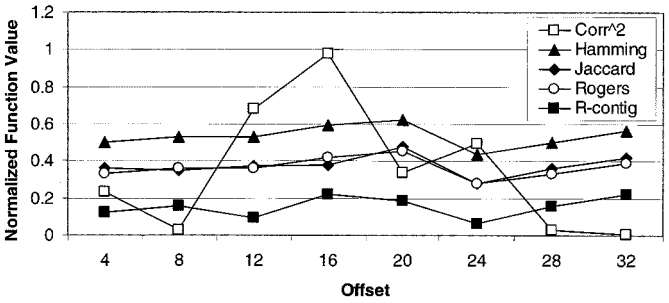


Fig. 8. Output values using a 4-B block comparison.

this folding of the density values about the origin produces invalid results because a value of $-1$, the result of two completely dissimilar strings, then becomes equal to an exact match. Other items to consider are the matching methodology and the sensitivity of the measures. The $r$-contiguous-bits measure is highly sensitive to bit changes near the middle of the string, while less sensitive at the outer edges. One bit flip in the middle can cut the measure's value in half, while an end bit change only decreases the measure by 1. This could be overcome if wrapping of the string is allowed. Finally, the matching methodology, whether block compare or sliding window, can produce very different results. By only comparing in successive $N$-bit blocks, information is lost (see Fig. 8). Calculations of the effect of block size on information loss show exactly this [36]. Most of our matching functions completely miss the exact match at position 14 because it is sandwiched between two successive 4-B blocks. The correlation coefficient comes close due to a false-positive match at position 16. It is hypothesized that the block compare methodology would only be useful in reduced instruction set computers, where instructions and data are aligned on predetermined boundaries. The chunk size would have to be exactly matched to the processor word size to be effective. Indeed, the entropy calculations in [36] show that local minima occur at instruction-size (4-B) boundaries. However, in complex instruction set computers (those running Microsoft DOS and Windows variants are host to the greatest number of viruses), instruction length is variable. Therefore, using a block compare strategy would miss important instruction and data structures. Conversely, for scanning packets, fields within the data stream are reserved for specific values. Therefore, block comparison in this problem domain makes sense.

## VIII. Immune-System Model Development

The components, processes, and results of the BIS show it to be an effective model for self-defense. It is desirable to construct a CDIS based on this model in order to overcome the reactive, nonadaptive, centralized, and monolithic nature of current computer-security solutions. However, the fundamental differences between biological and digital systems make a mapping between these domains difficult.

### A. Biological Immune System Features

In order to construct an effective isomorphism, the following features, functions, and organizing principles [37], [38] of the BIS must be understood.

1) *Parallel and Distributed:* The immune system is a massively parallel architecture with a diverse set of components. These components are distributed throughout the body and communicate through chemical signals.
2) *Multilayered:* No single mechanism offers complete immunity. Each layer operates independently, yet also in concert with all the other components, to provide defense-in-depth.
3) *Autonomous:* Each entity of the immune system operates under independent control. There is no central authority and hence no single point of failure. The multitude of independent agents work together resulting in the emergent behavior of the immune system.
4) *Imperfect Detection:* A detection event does not require an single exact match, but rather, the exceeding of an affinity threshold. Imprecise detectors allow for generality in the matching process, which further allows each detector to cover a larger subset of the nonself space.
5) *Safety:* The system contains checks-and-balances, such as costimulation or a second confirmation signal, and activation thresholds to ensure that detection errors are minimized.
6) *Diversity:* Diversity in the composition of each individual's immune system ensures that the entire population does not succumb to the same single pathogen. Additionally, each immune system cell only carries one form of detector. A large population of cells with a diverse set of receptor types enables the body to cover a large portion of the nonself space.
7) *Resource Optimization:* It is combinatorically expensive and too resource intensive to maintain a complete set of nonself detectors. Through the use of programmed cell death and cell division, the system maintains a random sampling of the search space at any one time.
8) *Self/Nonself Detection:* Through nonself receptor death and generation, the immune system has the ability to detect and respond to the presence of pathogens, even those that have not been encountered before.
9) *Selective Response:* After a detection, chemical signals and the identification method effectively classify the antigen. This determines the exact response to an infection.
10) *Memory:* Memory B cells enable the immune system to "remember" past infections and prime the system for an improved response upon later infections by the same or similar antigen.
11) *Adaptive:* The system evolves through clonal selection and hypermutation to improve the antigen recognition capabilities and therefore improve the overall system performance.

### B. Artificial Immune System Model

At a high level of abstraction, the main structures of the immune system map logically into information system entities (see Fig. 9). The BIS correlates to a CDIS, whose function is to detect and eliminate digitally malicious pathogens. These antigenic programs and network packets are made up of symbolic

TABLE II
BIOLOGICAL-TO-COMPUTATION DOMAIN MAPPING

| Immune System | Information System |
|---|---|
| Parallel & Distributed | Distributed system software utilizing data network communications. Detection and elimination activities operate in parallel. |
| Multi-layered | Multiple detector types monitor various input sources (email, file system, network packets, etc.). Policy guidance in order to implement barriers to initial infection (regular vaccinations, no executables in email, etc.). |
| Autonomous | A multiagent system of autonomous software agents. |
| Imperfect Detection | Detectors utilize partial string matching functions with an affinity threshold. |
| Safety | Costimulation through detection alarm validation to reduce false positive errors. |
| Diversity | Each detector node generates a statistically unique set of non-self detectors. |
| Resource Optimization | The detector set repertoire is continually resampled by reinitializing detector strings that are not activated within a certain time frame. |
| Self/Non-self Detection | Utilize the negative-selection algorithm to censor detector strings so that only non-self patterns remain. Employ these patterns through input source scanning. |
| Memory, Adaption | Retain detector strings that effectively match non-self. Upon multiple separate detector matches, only retain those with the highest affinity. |
| Selective Response | Eliminate malicious activity by the best means available, such as repair, deletion/replacement, quarantine, or port blocking. |

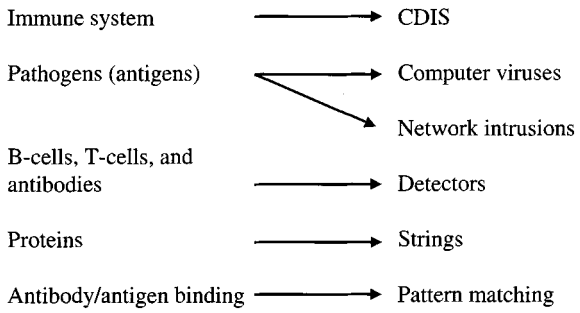


Fig. 9. Biological to computational domain top-level mapping.

string (i.e., bits, bytes, or words) patterns that detection algorithms search for by employing pattern-matching functions.

The previously identified BIS features, functions, and organizing principles are further decomposed into lower level information system entities and operations. The mapping between these functions and organizing principles can be seen in Table II. The autonomous, multilayered, and distributed features of the BIS suggest a distributed MAS utilizing a diverse array of agent detectors. These detectors maintain "antibody" search strings that are censored at creation via the negative-selection algorithm. The detectors are deployed with a pattern-matching function that produces a relative affinity based on the similarity of the antibody and antigen strings.

If a detector exceeds an affinity threshold, then it is activated. If multiple antibody strings are activated, "affinity maturation" is used to maintain only those detector strings that best match the malicious code. This process and the "programmed cell death" of nonactivated strings results in a continual searching of the nonself space along with a retention of only the best matching antibody strings. A match that exceeds the affinity threshold also requires a costimulation signal in order to reduce false-positive errors. A confirmed valid detection results in a selective response that utilizes the best means available, either repair, deletion, or quarantine for files or port blocking for intrusions. A repair can occur if an exact classification of the infecting virus can be made and a known "antidote" algorithm is available. Otherwise, the infected file must be deleted or immobilized (quarantined) in order to not pose a risk to the infected system or its neighbors.

### C. System Logical Hierarchy

The deployment of an agent-based CDIS should be distributed with redundant links and no centralized control in order to realize the fault tolerance and no-single-point-of-failure feature present in the BIS. However, a logical system hierarchy is required to apportion functional, management, and reporting tasks. These levels facilitate the dissemination of preventative information as well as the recognition and early suppression of computer-virus epidemics or coordinated network attacks (see Fig. 10). These communications links need to be encrypted and participants authenticated to ensure system integrity. This layered hierarchy is divided into the system, network, and local levels that map to a larger biological abstraction of populations, communities, and individuals (see Table III). The assignment of functionality to the three layers borrows from the structure and operation of the self-adaptive CVIS [6]. Similar layered architectures can be found in the Computer Health System [26] and Dasgupta's general ID framework [13].

1) System Level:
   a) provides health status of the community;
   b) identifies problems, durations, trends, and locations;

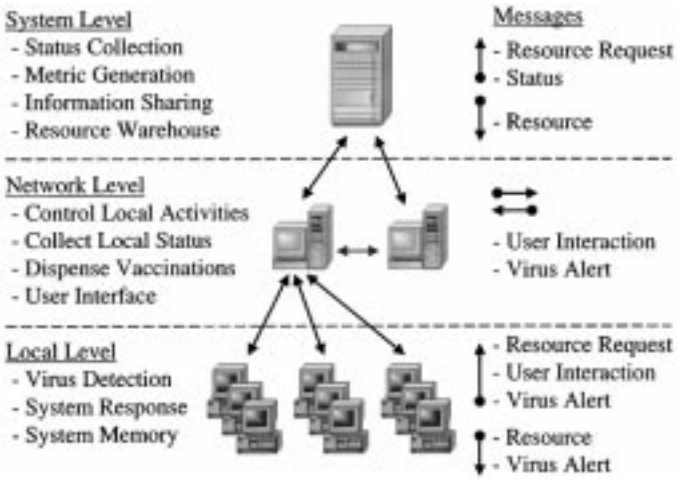


Fig. 10. Model logical hierarchy

TABLE III
SYSTEM HIERARCHY DOMAIN COMPARISON

| Level | Network | Biological |
|---|---|---|
| System | Internet | Population |
| Network | Subnet | Community |
| Local | IP | Individual |

c) promotes system health awareness by providing prevention information and sharing community status, thresholds, and vaccinations;
d) provides a global storehouse for memory detectors.

2) Network Level:
a) focuses on the local community of machines;
b) sets system priorities by controlling activation thresholds and system responses;
c) collects local system status;
d) reports local status to the system level;
e) dispenses vaccinations and preventative information.

3) Local Level:
a) responsible for detection, response, and memory;
b) implements innate and acquired immunity through self/nonself detection;
c) generates infection warnings;
d) implements local memory.

### D. Local Model

At the local level, detectors encompass the features of $B$ cells, $T$ cells, and antibodies into a unified detection entity. In order to reduce the overhead of maintaining multiple separate instances of detector objects each with a separate antigen receptor, each detector contains a set of detector strings. These strings are initially censored via negative selection and also have a finite lifetime, unless they are promoted to a memory "cell." False-positive errors are reduced through an affinity threshold and an external costimulation requirement. These processes infer the antibody scan string lifecycle model (see Fig. 11) introduced by [12] and expanded upon in [2] and [8].

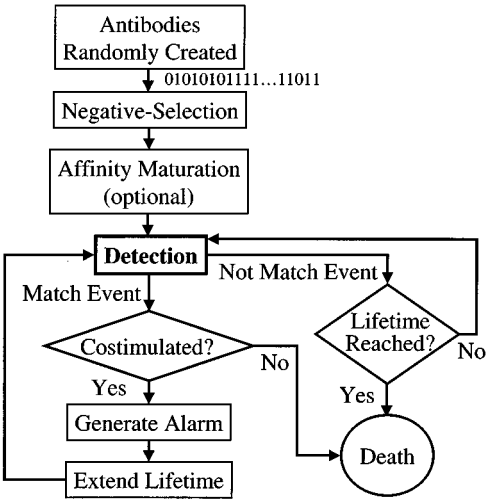


Fig. 11. Detector string lifecycle model.

## IX. DOMAIN-LEVEL DESIGN

The domain-level design involves defining agents and their interactions. This process is accomplished through use-case modeling. The biological, system hierarchy, and local models of operation imply a suite of operations performed by a federation of interacting agents. The identified operations of a computer-security AIS are the following.

1) Generate Nonself Strings:
a) the generator creates a nonself detector string;
b) the generator tests this string against all known self;
c) if a match on self occurs, the string is destroyed and a new string is generated. This process is repeated until no match occurs and the string graduates to an immature state;
d) if a detector string is set to memory type, the generator adds this string to nonvolatile storage;
e) the generator logs all actions performed.

2) Detect Foreign Bodies:
a) the detector opens the input source;
b) the detector performs pattern matching using one or more generated strings;
c) if a match occurs that exceeds the affinity threshold, the detector raises a warning and stores a pointer to the offending entity;
d) after a designated time period, if a detector string has not been elevated to a memory type, the detector destroys the detector string and signals the generator to generate a new one;
e) the detector logs all actions performed.

3) Monitor Warnings:
a) the monitor coordinates the activities of the local agents;
b) if a warning message is received, the monitor raises an alarm and signals the helper;
c) if an alarm is received from an adjacent monitor, the local monitor decreases the local affinity threshold;

d) the monitor communicates the local status to the controller;
e) the monitor logs all actions performed.

4) Costimulation:
a) if an alarm is raised, the helper reports the alarm and asks for costimulation;
b) if no costimulation is received or a negative costimulation is received, the helper signals the detector to destroy the detector string;
c) if costimulation is received, the helper signals the classifier and signals the detector to graduate the detector string from immature to memory state;
d) the helper logs all actions performed.

5) Classify:
a) the classifier gets the pointer to the malicious entity from the detector;
b) the classifier compares the data bits with known virus signatures or network intrusions;
c) if a match is found, the classifier signals the repairer;
d) if no match is found, the classifier signals the killer;
e) the classifier logs all actions performed.

6) Remove/Kill Foreign Body:
a) the killer notifies the helper that no known cure is available;
b) the killer asks the helper to confirm the deletion of the infected file, blocking of the port, or the shunning of an IP address range;
c) if a confirmation is received, the killer deletes the file or updates the firewall rules;
d) if no confirmation is received, the killer asks the helper to confirm the quarantining of the malicious code or the routing of traffic to a honey pot;
e) if no confirmation for quarantine is received, the killer warns the administrator of the presence of active malicious code on the system;
f) if confirmation for quarantine is received, the killer moves the infected file to a safe location and renders it unexecutable;
g) the killer logs all actions performed.

7) Repair:
a) the repairer notifies the administrator that a known cure is available;
b) the repairer asks the administrator to confirm the application of the repair;
c) if a confirmation is received, the repairer repairs the file, or resets the connection;
d) if no confirmation is recieved, the repairer asks the helper to confirm the quarantining of the malicious code;
e) if no confirmation for quarantine is received, the repairer warns the administrator of the presence of active malicious code or active network attack on the system;
f) if confirmation for quarantine is received, the repairer moves the infected file to a safe location and renders it unexecutable or routs network packets to a honey pot;
g) the repairer logs all actions performed.

8) System Control and Reporting:
a) the controller provides metrics to the administrator on system operation;
b) the controller provides the health status of the community;
c) the controller provides preventative information to the monitors;
d) the controller coordinates information passing between nonlocal monitors;
e) the controller logs all actions performed.

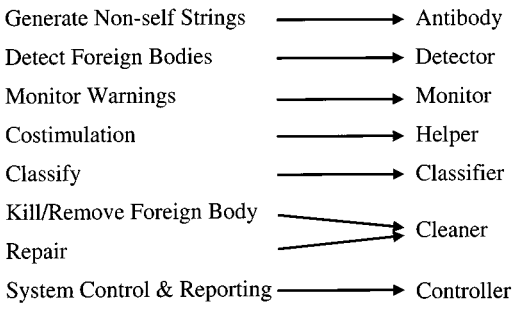


Fig. 12. Decomposition of use cases to agents.

The design of agent types is completed by decomposing the use cases into individual agents. A base set of seven agent types are identified and the mapping of use-cases to agents can be seen in Fig. 12.

The antibody agent encapsulates the generation and maintenance of search strings. The detector agent uses the services of multiple antibodies in order to scan an input string for malicious code or network intrusion signatures. The monitor controls the local area detection thresholds, communicates with the controller and other local monitors, and generates alarms to be acted upon by helper agents. Helpers perform the tasks of interfacing with the administrator, such as soliciting costimulation in order to overcome the problems of imperfect detector strings. Classifiers identify the exact infector or attack responsible and send the appropriate cleaner to fix the problem. Cleaners remove the virus or network attacks from the system using the best means available, repair, deletion, quarantine, or shunning. The definition of agent types concludes with assigning goals and services to the individual agents.

The agents with their goals and services can be seen in Table IV. The services provided by the agents are requested through interactions with other agents. These interactions are carried out by message passing "conversations."

### A. Agent Conversations

Agent "conversations" define possible interactions between agents [19]. Conversations are used by an agent to request the services of another in order to fulfill its goals. Through the coordinated use of each other's services, the CDIS as a whole is able to detect, identify, and remove malicious code from the system. The required coordination is accomplished through conversations.

TABLE IV
AGENTS, GOALS, AND SERVICES

| Agent | Goals | Services |
|---|---|---|
| Antibody | Generate, maintain, and store valid scan strings | Generate<br>Graduate to memory<br>Destroy scan string |
| Detector | Detect malicious code or network attacks at the input source | Scan input source<br>Receive vaccination<br>Update detection threshold<br>Destroy antibody string<br>Graduate antibody string to memory<br>Get pointer to input source |
| Monitor | Coordinate the actions of a local neighborhood of agents | Receive information from a Controller<br>Send, process, and receive alarm messages<br>Receive warning messages<br>Update detector detection thresholds |
| Helper | Communicate with the system user/administrator | Receive system information<br>Receive costimualtion<br>Receive action confirmation |
| Classifier | Implement the system response to an infection or attack | Identify malicious agent |
| Killer | Remove viral infections or network attacks | Delete malicious input |
| Repairer | Repair viral infections or network attacks | Repair malicious input |
| Controller | Coordinate global system operation<br>Generate system operation metrics | Receive monitor status messages |

TABLE V
AGENTS AND THEIR CONVERSATIONS

| Conversation | Initiator | Receiver | Description |
|---|---|---|---|
| cRaiseWarning | Detector | Monitor | Notify of a possible viral infection. |
| cUpdateThreshold | Monitor | Detector | An infection has occurred in an adjacent node, reduce the detection threshold to increase awareness. |
| cCostimulation | Monitor | Detector | A warning has been validated (not validated). |
| cVaccination | Controller<br>Monitor | Monitor<br>Detector | Vaccinate detectors with this string. |
| cSendMessage | Controller, Classifier, Monitor<br>Controller | Helper<br>Monitor | Notify the System level administrator with the attached text.<br>Notify the Network level administrator with the attached text. |
| cStatus | Controller<br>Monitor | Helper<br>Controller | Display system status to the administrator.<br>Send the Network level status to the System level. |
| cRaiseAlarm | Monitor | Helper | Ask for costimulation. |
| cConfirmation | Classifier | Helper | Ask for verification of virus removal actions. |
| cPassFilePointer | Detector | Classifier | Pass the Classifier the infection location. |

The conversations are developed from the use cases, where interagent interactions are described. Each interaction becomes a conversation or part of a more complex interaction. The use-case interactions generate the conversations shown in Table V.

### *B. System Design*

The system can be defined as a set of any number of different agent types [19]. The minimal set would be a monitor and a detector. However, a realistic system would include multiple instances of all the agent types running on distributed nodes.

The efficient mapping of agents to physical machines requires the considerations of parallel algorithm design. This is accomplished through two major components: 1) the identification of parallel components and 2) the mapping of tasks to processors to minimize communication [39]. The division of tasks into those that can operate concurrently occurs as part of the agent decomposition. The second part of a good parallel agent deployment is the consideration of communications costs. For example, file detectors need to be local to their file system to avoid passing large amounts of data (conceivably the whole disk) across a network. The network intrusion detectors would need to be placed on a network border machine, probably just behind a firewall. In general, attacks or infections are rare, so detectors run locally and send messages to their associated monitors in order to minimize network traffic. Due to input–output (I/O) considerations, the classifiers and killers should also be located local to the input source. These agents perform file operations or update firewall settings, which can induce considerable network loading if done remotely.

A major system consideration is the need for low resource overhead. The CDIS should be unobtrusive to the user. Because infections and detections are rare, helpers, killers, and cleaners are not used often. Therefore, in order to not waste central processing unit (CPU) cycles or memory on busy waiting, these agents are instantiated only when the need arises. Helpers only send messages and perform costimulation. It is logical that they be colocated with the monitor for simplified user interaction. All these considerations are embodied in an example physical deployment diagram (see Fig. 13).

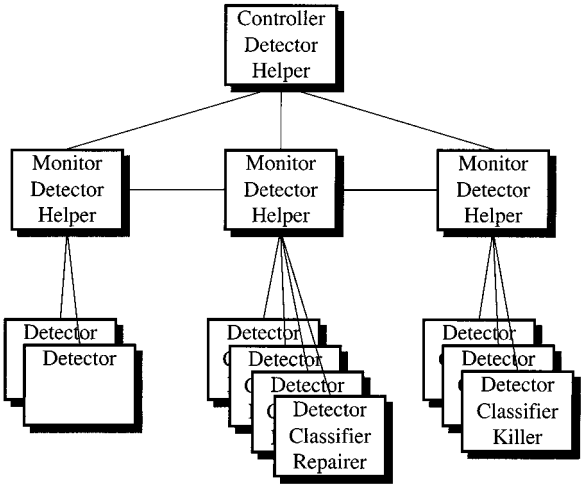


Fig. 13. Agent deployment diagram.

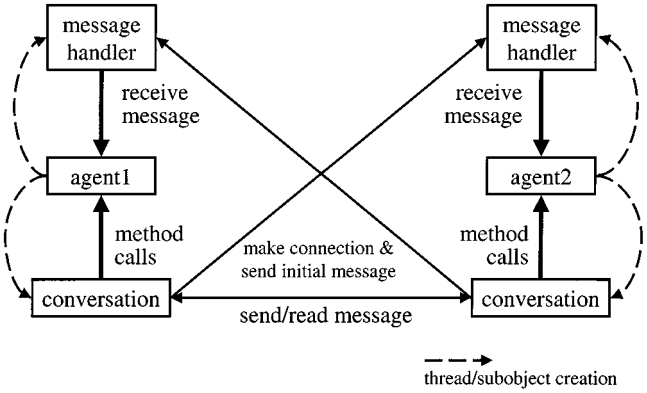


Fig. 14. AgentMOM operation.

## X. Agent Communications

This CDIS is designed as a multiagent system. These autonomous system entities collaborate with each other in order to produce an immune system behavior. This collaboration, which is inherent in distributed multiagent systems, requires the use of a network backbone and a communications software layer. The Java language was chosen for this project because it was designed to operate over networks and, hence, provides comprehensive network communication support. However, low-level TCP/IP socket construction, manipulation, and optimization is not the goal of this research. In order to develop the distributed agent-based CDIS prototype, a communications library that abstracts away the low-level details of network communication is desired. There are many approaches to this problem of distributed computing, including shared memory, message passing, distributed objects, and agent development kits.

The ideal communications library would provide an efficient abstraction above the low-level implementation issues while supporting the needs of agent collaboration, the immune system model, and the desire for fast prototyping. Foremost is the need for low startup and transmission overhead. The bottleneck in many distributed applications is the communication time. In order to further minimize communication costs or to not undo the efforts of effective agent decomposition, an efficient communications library is required.

The needs of the agent design require the use of one-to-one and one-to-many send routines. For example, vaccinations should be broadcast to all the appropriate detector types, while virus detection warnings need only be sent from a single AV detector to a monitor agent. A messaging system that only provides one-to-one capabilities could be used by making multiple sends to a list of recipients, but this would be less efficient, especially in a local area network (LAN) environment, where packets are broadcast to all nodes anyway.

On the receive side, asynchronous messaging is desired. An infection and later detection of the malicious code occurs with a relatively low frequency. Responses to an infection are driven from the detection event. Therefore, an asynchronous event-driven messaging system is desired. Next, agents can pass these messages between each other, within possibly multiple separate conversations. A communication layer that supports multiple channels over a single connection would be ideal.

The system is designed as a collaborating federation of agents. These agents could conceivably join or leave the group at any time; for instance, if workstations were turned off at the end of the day. For this reason, it is desired that the communications library supports the ability to join and separate from the system, or subscribe and unsubscribe to message-passing channels.

Finally, with an eye to the future, the system should be able to incorporate a security layer. In order to make a fielded system resistant to infiltration or spoofing, encryption of messages and the authentication of agents would be required. Such features are beyond the scope of this prototype, but would be necessary in an actual deployment.

Along with this diverse set of functional requirements, a communications layer that is easy to use and understand is desired. This facilitates later understanding and expansion of the design. Two communications systems are selected to implementation the CDIS multiagent communications. The CDIS communications infrastructure and agent messaging components are realized by combining the strengths of message-oriented middleware (MOM) and the Java Shared Data Toolkit (JSDT).

### A. AgentMOM

AgentMOM is a communications framework developed by the AFIT Agent Research Group [40]. It is designed to explicitly implement the communications required in MASs engineering-designed architectures. Although agentMOM is termed as a MOM for agents, it is actually devoid of middleware services commonly associated with MOMs, such as automatic message routing or queuing. However, it has been proven effective for implementing agent conversations.

Agents utilizing this framework implement two components: 1) the message handler and 2) the conversation (see Fig. 14). Agent communication occurs via conversations in a multiagent systems engineering environment. When an agent wants to collaborate, it begins a conversation as a separate thread. The initial message is sent across a socket connection to the recipient's message handler. The message handler monitors a local port for incoming messages, which it passes on to an agent's

*receiveMessage* method. The *receiveMessage* routine processes that message and, if appropriate, begins the other side of the conversation in a separate thread. After that initial contact, the conversation is handled by the two conversation threads. Utilizing threads for conversations eliminates agent busy waiting during blocking communication calls.

Messages in this framework are sent as the content in peer-to-peer conversations. AgentMOM does not directly support one-to-many multicast messaging. This would have to be simulated by using multiple one-to-one calls. The agentMOM architecture utilizes asynchronous event-driven messages and multichannel messaging is accommodated via multiple conversations all running as separate threads. AgentMOM does not use subscription-based channels, instead conversations are initiated and torn down as required. This is potentially more efficient if conversations are infrequent, as is the case with those initiated on virus detection. This lack of a subscription service also alludes to agentMOM's low level of abstraction. AgentMOM requires the programmer to specify socket addresses and ports. However, because operations are at this level, performance gains can be realized through tailoring of the operations to the exact problem domain. Additionally, this facilitates the addition of extra functionality. For instance, at this level, security is not implicitly offered; however, the socket constructor could easily be replaced by one that implements secure socket layer (SSL).

AgentMOM offers a medium-level abstraction for agent communication. Instead of middleware services, as the name implies, the library provides base classes and functionality to the individual agents. AgentMOM partially defines the structure of the agents themselves, not just the communications mechanisms. For instance, the passing of received messages to an agent's *receiveMessage* method is specified. An additional benefit of the lower abstraction is performance improvements gained by a reduced number of object layers as well as the capability for implementation tailoring. AgentMOM provides an ideal architecture and agent functional description for implementing conversations, but it lacks some communications services desired for our system. Therefore, its overall architecture is combined with the JSDT.

### *B. Java Shared Data Toolkit*

The JSDT is a communications library that is designed to support collaborative applications [41]. This set of classes provides an abstraction above the basic networking functionality to offer communication sessions between objects, with each session capable of supporting multiple separate data channels. The low-level networking communication can utilize sockets, hypertext transfer protocol (HTTP), light-weight reliable multicast package, or remote method invocation for its basic connection. The exact method can be specified by the programmer during session creation. Since most of these protocols are built on sockets, it makes sense to utilize the basic socket for efficiency.

This architecture can efficiently support multicast messages with point-to-point being a special case. There is also support for both synchronous and asynchronous message delivery, with the latter being the default. In the asynchronous mode, a channel consumer's *dataReceived* method is called when a message arrives, thereby providing an event driven operational model. Channel consumers indicate their interest in a particular session:channel combination by subscribing to it. Additionally, the library supports managed sessions. A session manager can invite clients to join a session channel or even expel them from an existing connection. Inherent to a managed session is a security layer consisting of a challenge/reply authentication between the manager and the joining client. Additional security can be added by utilizing a SSL instead of regular, unsecure socket connections. Utilizing this capability is as simple as adding two source code lines at the beginning of a JSDT application.

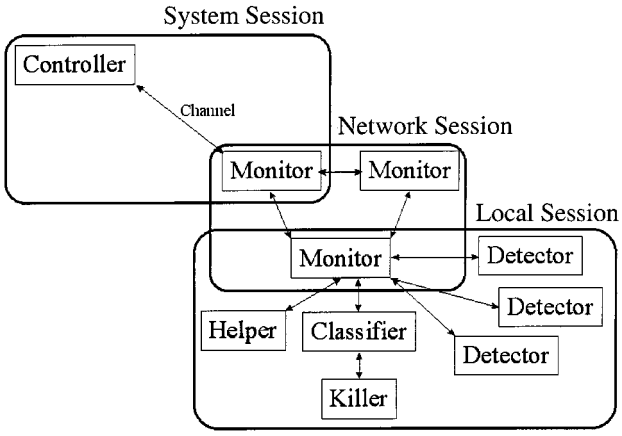


Fig. 15. Session-level logical view.

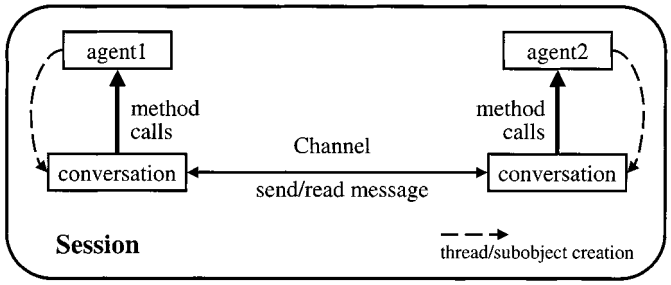


Fig. 16. Channel-level logical view.

### *C. CVIS Communications Design*

The JSDT constructs are combined with the agentMOM architecture to provide a hierarchical communications network that supports the system, network, and local CVIS levels (see Section VIII-C). By utilizing the JSDT session constructs, the implementation can create multiple sessions to logically isolate conversations at the appropriate level (see Fig. 15).

The system-level session encompasses the regional (or global) controller agents and their assigned network monitors. At the network level, various sessions connect local Monitors in order to pass on virus epidemic or large-scale network attack warning messages. Finally, many local sessions connect the monitors to their assigned agents. Within these sessions, multiple channels are created in order to carry on interagent communications (see Fig. 16). Each of these conversations is implemented as a separate thread.

TABLE VI
TEST INPUTS

| Number | Description | Purpose |
|---|---|---|
| 1 | Self - Randomly generated<br>Non-self - Randomly generated | Does CDIS operate in a large search space? |
| 2 | Self - Application program suite<br>Non-self - EICAR test program | Does CDIS operate correctly in a real world environment? |
| 3 | Self - Application program suite<br>Non-self - TIMID virus | Comparison of this CDIS to other research. |
| 4 | Self - LL Training set<br>Non-self - Nessus probes | Does CDIS operate correctly in a real world environment? |

The operation of conversations are based on agentMOM constructs. However, because JSDT channels are used, the message handler is unnecessary and agents can use the session members list as a naming service to invite other agents to join a conversation. Conversations are implemented as separate threads in order to allow multiple concurrent conversations within a single agent. This also enables agents to better enforce accountability in multicast type conversations. The following is the agent conversation process.

1) The agent joins a session.
2) The agent looks up the other agents in their shared session to begin a conversation with one (or more) of them.
3) The agent creates a new conversation thread.
4) The conversation creates a new managed channel within the session.
5) The conversation invites the other agent to join the channel.
6) The agents converse by passing messages back and forth.
7) The initiating conversation thread expels the other agent from the channel.
8) The conversation thread closes the channel.

By utilizing the capabilities of the JSDT, combined with the overall architecture of agentMOM, a solution that meets all of the system requirements is obtained. Furthermore, this implementation elegantly captures the agent conversation paradigm.

## XI. TEST PLAN

The purpose of the system experiments is to understand the performance implications of the CDIS agent components. The objectives of system experimentation are to gain insight into the efficiency and effectiveness of this prototype system.

### A. Influential Variables

The influential variables are those items that may be controlled or uncontrolled and have an effect on system performance [42]. Each of these involve engineering tradeoffs in system design and they may not be independent. The variables and some of their effects are listed below.

1) *Affinity threshold:* the level of detection required to raise an alarm. The threshold can affect Type I and Type II error rates.
2) *Antibody length:* the number of bytes in an antibody string. String length can affect memory usage and antibody effectiveness.
3) *Number of antibodies in a detector:* the number of antibody strings in each detector can affect the probability of detection.
4) *Contents of self and nonself:* the degree to which the self/nonself data appropriately represents all possibilities can affect the Type I and II error rates. Lack of specificity, where some self data is indistinguishable from nonself, can lead to an autoimmune reaction.
5) *Length of self and nonself:* the size of the data sets, whether the total file system size or the total number of allowed network requests. Data set size affects scan time and negative-selection time.

### B. Test Inputs

There are three basic sources of test problems: 1) those that arise naturally in practice; 2) ones that are specially constructed to test a particular aspect of the code; and 3) randomly generated problems [42]. Additionally, it is desirable to test against a common industry benchmark. Our testing uses all four.

The test problems for a CDIS are sets of self and nonself strings. In order to test the operation of the antibodies, some are assigned to predetermined values. To test the CDIS's ability to function in a large search space, randomly generated sequences are used. Finally, the system is tested against actual user programs, viruses, and captured network traffic in order to understand the system's applicability to the real-world problem domains. The complete set of test inputs can be seen in Table VI.

Of particular use for the real-world computer-virus problem set is the European Institute for Computer Antivirus Research (EICAR) standard AV test file [43]. This file contains of a set of 68 B of ASCII printable characters. The purpose of the file is to provide a safe target for testing the operation of AV software. The file is easy to use and noninfecting. It is an executable file that only prints the message *EICAR-STANDARD-ANTI-VIRUS-TEST-FILE!*. Most commercial AV software products have scan strings that recognize the EICAR test pattern.

One of the key reasons for utilizing an immune system model of operation is to recognize as of yet unknown viruses. Therefore, a modified version of the EICAR test string is used as a new unknown "virus." For this purpose, EICAR was modified so that it now prints *Paul Harmer-s test Virus XxXxXxXx!!* instead. This new noninfecting strain of EICAR goes undetected by Norton AntiVirus (NAV).

Testing the system against a common industry benchmark is desired in order to compare the efficiency and effectiveness of the proposed CDIS against other solutions. Unfortunately, there

TABLE VII
TEST CASES

| Number | Input | Measurements/Desired Output | Variable |
|---|---|---|---|
| 1 | 2 | Can known and unknown viruses be detected? | EICAR vs. EICARnew |
| 2 | 1 | Type I and Type II error rate vs. number of antibodies | Number of antibodies |
| 3 | 1 | Type I and Type II error rate vs. antibody length | Antibody length |
| 4 | 1 | Type I and Type II error rate vs. file system size | File system size |
| 5 | 1 | Type I and Type II error rate vs. threshold level | Threshold level |
| 6 | 1 | Scan time vs. Number of antibodies | Number of antibodies |
| 7 | 1 | Scan time vs. antibody length | Antibody length |
| 8 | 1 | Scan time vs. file system size | File system size |
| 9 | 1 | Negative-selection time vs. number of antibodies | Number of antibodies |
| 10 | 1 | Negative-selection time vs. antibody length | Antibody length |
| 11 | 1 | Negative-selection time vs. size of self | Size of self |
| 12 | 3 | Number of immature antibodies required to generate a number of antibodies | Number of antibodies |
| 13 | 4 | Can known and unknown network attacks be detected? | LL Data with Nessus Probes |

does not exist such a baseline. This prototype represents one of the first CDISs constructed that addresses the virus problem. However, tests were performed in [30] to validate their $r$-contiguous-bits theoretical derivations against actual data. For these tests, the TIMID virus [24] was used to infect COM files. We also chose to use TIMID for testing.

TIMID is a simple file infecting virus [24]. It only infects one file on each execution. Its targets are COM files residing only in TIMID's local directory. It does not hop across directory structures. Additionally, TIMID has the nice feature of outputting the name of its victim.

TIMID is an appending file infector that adds 5 B to the beginning of a file and an additional 300 to the end. No stealth capabilities are employed, so victim files sizes can be seen to grow by 305 B, along with an appropriate file date alteration. All these features make TIMID an excellent test subject because it can be controlled and its effects are known. Furthermore, it is a commonly known virus that can be detected and removed by all current AV suites. TIMID, EICAR, and the generated problem sets become inputs to the test cases.

For real-world network intrusion testing, a combination of captured network packets and probe seeding is used. At first, randomly-generated packet sets were utilized, but they did not prove very useful. It was thought that the use of generated data would make evaluation of the algorithm easier; however, even when limiting the generation range to a few fields and a fairly small range of parameters, the highly random distribution of the data prevented achieving consistent test results. Generating a more structured data set would have solved some of the problems, but would have had to be done with extreme care so as to model actual traffic in a useful way. Therefore, all testing on the ID aspect of CDIS was completed using captured data.

MIT's Lincoln Laboratories (LL) completed an ID evaluation for which they created a corpus of ID data [44]. The LL data was designed and generated with this type of research in mind and its use made the evaluation of CDIS feasible. The entire ID corpus contains both network and host sensor logs, as well as file system dumps and directory listings (among other things) for a national Air Force base. For this effort, only a small subset of the network data captured outside the test firewall was used. Negative selection was performed using only the LL training data—data without intrusion attempts. For scanning, a small number of malignant packets generated by the Nessus security analysis tool were added [45].

### C. Test Cases

The test cases are designed to gather effectiveness or efficiency data. In each test case, either an influential variable is changed or a static system property is measured to understand system performance. The test cases are enumerated in Table VII.

Each test is run five times. Regarding the number of parametric and nonparametric experiments required in generating viable statistics, it depends upon the specific experiment and desired confidence interval ([46, p. 431]). Also, the experimental goals must be considered, which drives the selection of experimental parameter values as well as the performance metrics. Metrics are of the effectiveness category (such as solution quality) or efficient category (such as computational effort or algorithm efficiency) [47]. The intent of the efforts reported here are to give an appreciation of possible qualitative statements with little emphasis on efficiency since the goal is focused on feasibility. As to the number of experiments, an increasing number is always required until a distribution of results is achieved resulting in an underlying statistical model. This model, based upon quantitative data, can then be employed to make qualified statements with a high level of confidence. Our limited experiments give a consistent, but limited view of performance. In the future, more extensive statistical experiments are required using different parameter values and associated sensitivity analysis in order to reflect performance means and variances and confidence levels across a wide range of realistic intrusion test data. Such analysis could achieve a much higher level of confidence in the utility of our suggested virus and ID process. It is not the intent of

our current efforts to show a high level of confidence in the parameter space because of the extensive and abnormal testing required over incomplete intrusion data. However, we do desire to convince the reader of the feasibility of the approach, which we hope we have done from an unbiased perspective.

### D. Testing Platform

The CDIS is tested on the AFIT bimodal cluster (ABC) pile of personal computers (PCs). This is a heterogeneous system consisting of 22 variously configured Pentium II and Pentium III CPUs connected by a fat tree gigabit and 100baseT switched Ethernet backbone. Each machine within the cluster is dual bootable as a Windows2000 or Red Hat Linux system. All systems are booted as Windows2000 systems to reflect the most common virus target platform and current Air Force server standards. All network intrusion testing was completed within a segregated laboratory. The network intrusion agent was hosted alongside the Objectivity 5.2 database on a single Windows2000 computer. Tests on the detector agents were performed using simple system architectures consisting of one controller, one monitor, and one detector. Testing of system scaleability and agent performance were conducted utilizing one controller and two monitors, each with two detectors. The testing of the ID and AV detectors were performed separately, although each detector-type integrates well into the larger hierarchy.

The ABC is a closed environment that is representative of a PC LAN network, the target implementation platform of the CDIS. Increasing the test platform to include Win95 or even Solaris machines would be a good test suite for understanding the performance of the CDIS in the enterprise environment and the effectiveness of the network intrusion detector in actual deployment conditions. Exploration into a more robust, fieldable CDIS platform is left for future research.

## XII. Experimental Results and Analysis

### A. Negative-Selection Time

The negative-selection algorithm represents an investment that the system must make in order to remove the possibility of false-positive errors (see Section VI-A). The current algorithm sequentially checks each antibody against all bytes in the known self space. For the virus detection antibodies, this requires adding each byte from self to the sliding window and then comparing each antibody bit by bit or checking each packet within the known self database in the case of ID. negative selection is performed after each antibody string is randomly generated. If a match on self occurs, the antibody is regenerated and retested from the beginning of self. Alternatively, a pool of previously generated antibodies could be run through the negative-selection process together. Theoretically, the negative-selection time should grow linearly with respect to the number of antibodies, the length of each antibody, and the size of known self. This is because each byte in every AV antibody must be checked against every self byte.

| | |
|---|---|
| $N$ | Number of antibodies. |
| $L$ | Antibody length (bytes). |
| $S$ | Size of self (bytes). |
| $O(L)$ | Sliding window shuffle. |
| $O(8L)$ | Bit compare. |
| $O(N(S(L+8L))) = O(NSL)$ | Negative selection. |

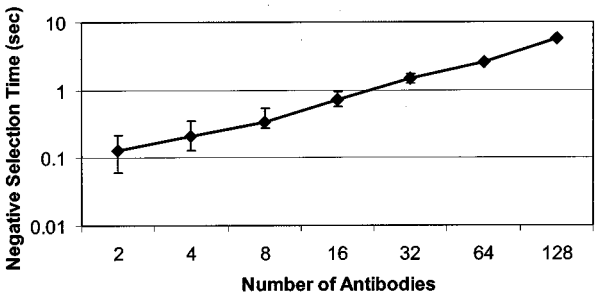


Fig. 17. Computer-virus negative-selection time versus the number of antibodies per detector.

This linearity is somewhat deceiving however. For an *a priori* defined number of initial antibodies, it is true. It can be considered a lower bound, but the number of antibodies required to protect a system with a chosen probability of detection grows exponentially with the size of self [30]. Therefore, the number of antibodies $N$ would be expected to grow exponentially with the size of self.

These theoretical results are also representative of the ID scanner if it were to be deployed against a log file. However, our tests with packet scanning utilize a database to enable better experimentation. The negative-selection time and the scan time are, therefore, dependent upon the database query algorithm.

The experimental results accurately follow the expected theory as the time tends to double as the number of antibodies are doubled in the AV detector (see Fig. 17). The network ID agent performance is similar to the AV agent (see Fig. 18). These tests were accomplished using the Rogers and Tanimoto matching rule with a 0.7 affinity threshold against 1 K of randomly generated application self bytes (for AV) or with 10-K self packets (for ID) from the LL's set. For these tests, a candidate antibody pool was not generated, so the variations in the negative-selection times are due to matches on self. A match during this process requires the system to regenerate a new antibody and then compare it against the entire self set. Because antibody creation is random, the negative-selection process introduces random variation in the negative-selection times. It is hypothesized that the probability of a match is low for this 1 K set of self so that the linear equation holds. With a higher probability of match, an exponential trend would result.

Fig. 19 depicts the effects of antibody length on the negative-selection time for the AV layer. The ID antibodies use a predefined length of 320 bits. Therefore, length is not a factor. But, for the AV agent, at lengths greater than 2 B, the negative-selection time grows at an almost imperceptible rate that is superlinear. The change in length also induces very little variance in the negative-selection time. However, a 2-B string produces very dramatic increases in the censoring time with accompanying wide variance between runs. This result is consistent with the comprehensive theory developed in [30]. The ob-

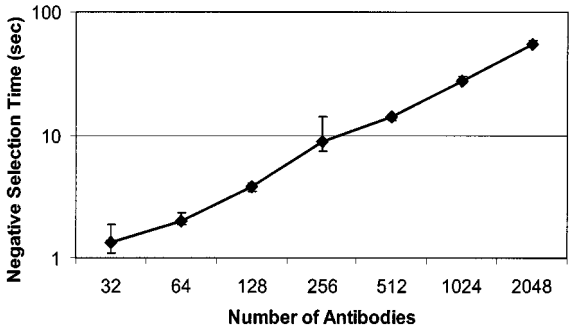


Fig. 18. ID negative-selection time versus the number of antibodies per detector.

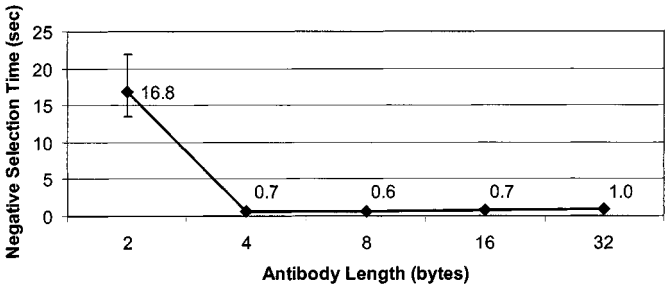


Fig. 19. Computer-virus negative-selection time versus antibody length.

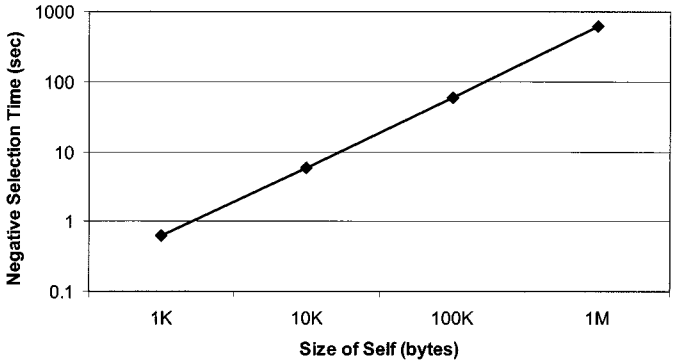


Fig. 20. Computer-virus negative-selection time versus the size of self.

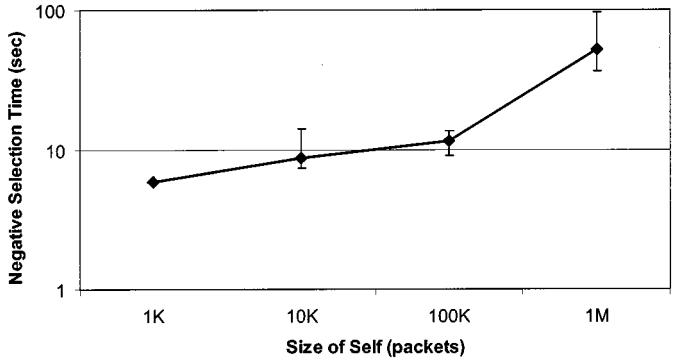


Fig. 21. ID negative-selection time versus the size of self.

served 25-fold increase in the negative-selection time is due to the increasing generality of a 2-B string, or the increased probability of a match during negative selection. Somewhere between 2 and 4 B there is a sensitivity point, before which a very large number of matches on self occurs. The result is a much larger negative-selection time in order to find the required number of 2-B combinations that do not match self. The 4-B antibody falls at the beginning of this trend. It has a negative-selection time that is slightly greater than the 8-B case.

The previous tests utilized 1 K of self bytes in order to censor the antibodies. Fig. 20 shows the effect of the length of self on the AV antibody's negative-selection time. The comprehensive theory [30] predicts that an exponential relationship would result; however, a linear relationship is observed with very little variation in the experimental times. This variation is visualized as almost imperceptible error bars in Fig. 20. It hypothesized that the observed linearity is due to a low probability of match with the Rogers and Tanimoto rule. More testing and theoretical calculations are required to fully understand its relationship in the comprehensive theory, but the testing does give us insight into the system efficiency. The system produces 16 8-B antibodies against 1 MB of known self in approximately 10 min. Using the more general 2-B antibodies could cost over 25 times that on average.

The generation of correctly censored antibodies produces the core components for virus detection by an AIS. File systems are large and growing. For example, an 8-GB hard drive is considered small for commercially produced PC's. The current performance of this system (assuming the linear relationship is maintained) would produce 128 4-B antibodies against 8 GB of self in 1.45 years! Clearly, this is too long to be practical and any reduction in antibody length or detection threshold would only increase this time. Algorithmic and implementation improvements are required to reduce the negative-selection time to a usable duration.

Fig. 18 shows a superlinear, possibly exponential, increase in the cost of performing negative-selection on antibodies for network ID. The upswing is likely caused by a combination of two factors. One factor is inefficiency in the database design, resulting in nonlinear increases in query service rates for larger table sizes. This factor is being ameliorated through the development of optimized data structures and memory-resident storage to eliminate disk accesses. Some polynomial growth cannot be eliminated, however, due to the requirement to find the intersection of the responses to multiple queries. The other factor is related to the diversity of the self data. If the self data is highly homogenous, then additional data points would have minimal effect on antibody generation: antibodies describe regions and a small number of self data points would be enough to eliminate bad antibodies. However, if the self data are relatively scattered, then as the number of data points increases, the probability of generating an antibody that has to be discarded also increases. Larger data sets imply longer comparison times, exacerbated by increased probability of having to regenerate antibodies; hence, a superlinear performance curve results. This phenomenon is endemic to all inductive learning processes; a useful area of future research would be to determine the homogeneity of the LL and real-world training data sets, thereby providing a basis to address the question of "How many samples are enough?" so we can optimize the training process.

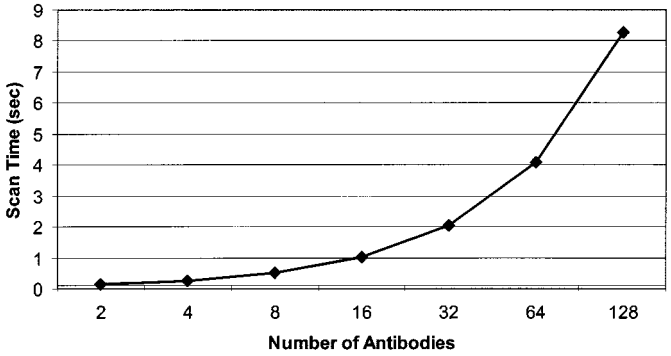


Fig. 22. Computer-virus scan time versus the number of antibodies per detector.

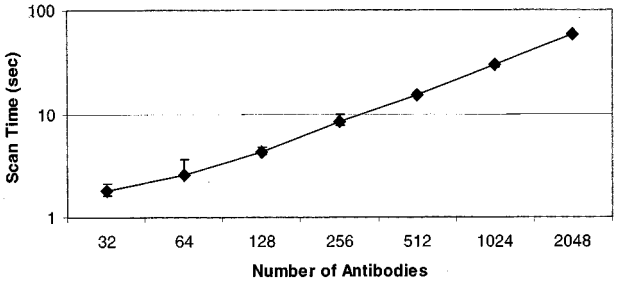


Fig. 23. ID scan time versus the number of antibodies per detector.

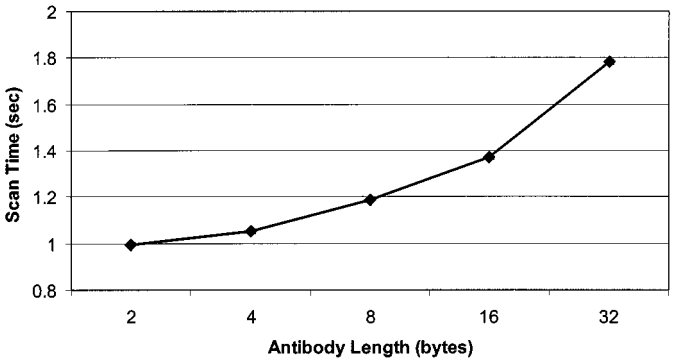


Fig. 24. Computer-virus scan time versus antibody length.

### B. Scan Time

The scan operation represents the heart of the AIS detection process. Ideally, the viral scanning algorithm should run as quickly as possible in order to be unobtrusive to the user. For an IDS, the algorithm should also be high-speed so that it can keep up with the flow of data coming over the line. If not, the IDS either becomes a bottleneck or ineffective as some packets are skipped. Current systems often fall short of this goal, especially as data sizes and networking speeds continually increase.

Theoretically, the scan time of this system is directly proportional to the amount of data being scanned, the number of antibodies, and the antibody length. For detecting file viruses, the scanning algorithm must read in each byte of the file system, add it to the sliding window, and then compare the window against the antibody string bit- by-bit.

| | |
|---|---|
| $N$ | Number of antibodies. |
| $L$ | Antibody length (bytes). |
| $X$ | Size of file system (bytes). |
| $O(L)$ | Sliding window shuffle. |
| $O(8L)$ | Bit compare. |
| $O(X(L+N(8L)))=O(NXL)$ | Scan time. |

This analysis is somewhat irrelevant to an actual deployed packed-based ID scanner because packets would be read as they flow by on the wire. In our experiments, however, we utilize a database loaded with actual captured packets. Here again, like the negative-selection time, it is expected to be driven by the database query algorithm; the only difference being that matches result in a detection alarm instead of an antibody regeneration and certification process. Therefore, the IDS average scan times should be slightly shorter than the negative-selection times, with a smaller variance.

The experimental results hold true to theoretical expectations. The number of antibodies in a detector directly affects the scan time (see Figs. 22 and 23). As the number of AV antibodies are doubled, the scan time is also doubled. Compared to the negative-selection algorithm (see Section XII-A, Fig. 17), scanning produces negligible variations in execution time. During scanning, if a match occurs, the offending file and its bound antibody are added to a list. This requires no regeneration or rescanning as in negative selection. Therefore, scan times are almost constant between runs.

Likewise, the ID times mirror those produced by negative selection, with reduced variations. The result is a much smoother growth in the time curve. Unfortunately, the time advantages gained by utilizing the database for testing cannot be directly related to ID scanning directly from the wire. Because the database can perform the indexing operation ahead of time, considerable searching advantages are gained. However, realistic growth rates are seen, along with effective operation. Future research will replace the database I/O component with wire-sniffing operations in order to fully assess the system usability.

The network attack antibodies are a fixed length, but the AV antibodies need not be so. Antibody length was expected to affect AV scan time linearly; however, the results indicate a slight superlinear trend (see Fig. 24). We hypothesize that this is due to the Java implementation of bit comparisons. In general, these results give a few specific long antibodies an advantage over many short strings. Longer strings can be used with only small performance ramifications. The long antibodies also result in relatively short negative-selection times due to their specificity (see Section XII-A, Fig. 19). The tradeoff is in the ability to effectively search the larger space created by utilizing specialized detectors.

The experimental results also parallel theory with respect to file system growth (see Fig. 25). Increasing the file system size ten-fold also increases the scan time by a factor of ten. The 2-MB file system is scanned in 19.5 min. By extrapolation, an 8-GB file system with 128 4-B antibodies would be expected to take 1.05 years to scan.

Scanning is faster than negative selection because files only need to be opened once and compared against all antibodies (see Section XII-A). The current negative-selection algorithm

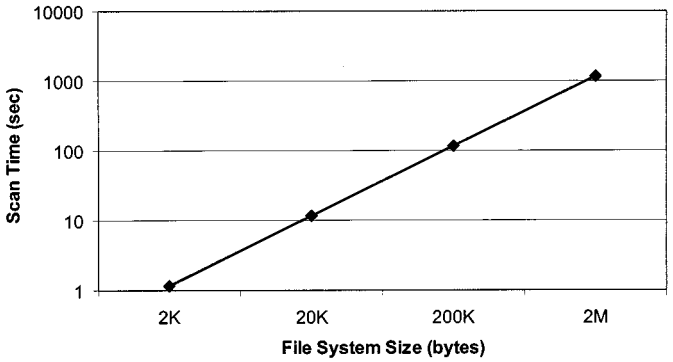


Fig. 25. Computer-virus scan time versus file system size.

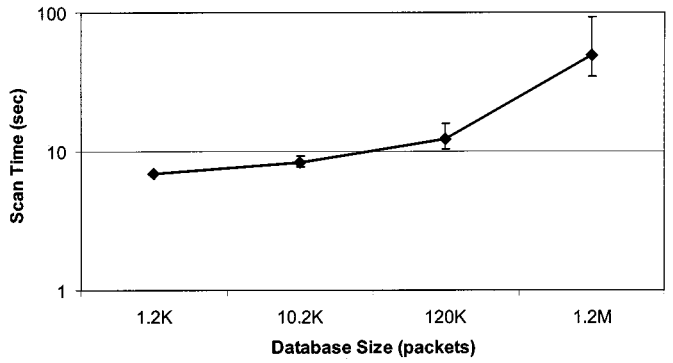


Fig. 26. ID scan time versus input database size.

requires opening every file in the system once for each antibody. Therefore, the I/O system overhead is incurred multiple times unnecessarily. This observation gives insight into possible algorithmic improvements for negative selection. As with negative selection, the scan time is too long to be of practical use. Algorithmic and implementation improvements are needed to make the system usable.

The ID scan times for an increasing number of packets can be seen in Fig. 26. The results are slightly better than those for negative selection (see Fig. 21). Since antibodies are not being regenerated, the probabilistic and diversity arguments posed earlier do not apply. This would appear to indicate that the dominant factor in the superlinear growth of both figures is indeed an artifact of the database. As mentioned earlier, this factor is being addressed in ongoing research.

### *C. Error Rate*

The system's error rates reflect its ability to detect self and nonself appropriately (see Section V). The false-positive rate should always be zero. This is ensured in advance by the negative-selection algorithm. By initially censoring strings against self, no future self-matches should occur (assuming a static definition of self). The false-negative rate should also ideally be zero. Any percentage higher than this indicates the system's relative inability to detect the presence of nonself. This rate can fluctuate dramatically because of the stochastic nature of the problem. The antibody strings are randomly generated, as are the appearance of viral infections or intrusion attempts. So, the system parameters, such as antibody length, the number of antibodies, and the detection threshold, must all be tuned in

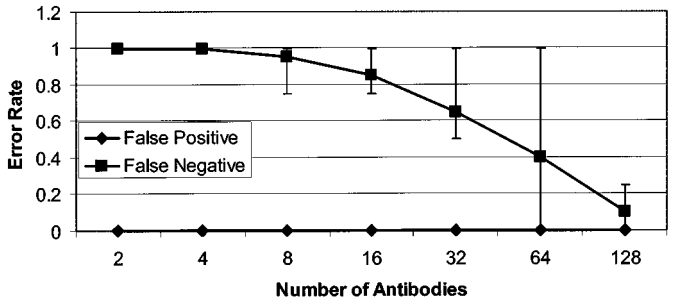


Fig. 27. Error rates versus the number of antibodies per computer-virus detector.

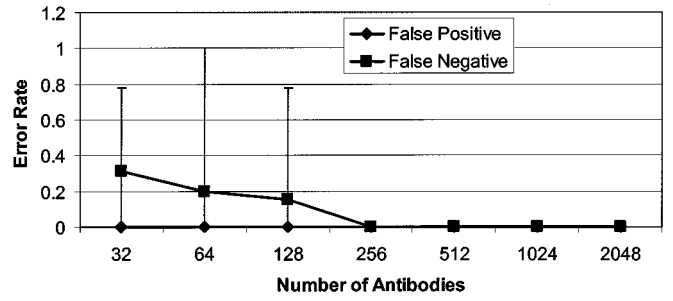


Fig. 28. Error rates versus the number of antibodies per intrusion detector.

order to minimize the false-negative rate. The comprehensive theory developed in [30] is able to predict the probability of failure (false-negative rate) given a matching rule probability of match. Experimental results support this theory. However, this theory has not yet been adapted to the Rogers and Tanimoto matching rule that the CDIS utilizes. Therefore, experiments are performed to understand the feasibility and effectiveness of this matching rule for AV and IDSs. For these feasibility tests, each AV or ID detector contains a set of antibody strings. Each test is run five times and an average error rate is determined.

Increasing the number of AV antibodies generally decreases the average false-negative rate (see Fig. 27) for the antibody sets. This test utilizes 4-B antibodies and a 0.7 detection threshold against 1 K of randomly generated nonself bytes. The error bars indicate the maximum and minimum values to understand the complete range of effectiveness for the five test runs. Due to the probabilistic nature of the problem, even 64 antibodies can fail to find nonself the same as a single antibody. Conversely, the best run of 64 or greater number of antibody strings found all the nonself files. In order to generate a consistently low error rate, 128 or more antibodies are required per detector. However, this results in higher negative-selection and scan times. A tradeoff between speed and coverage must be made.

The ID agent is effective at higher antibody counts, but it does not exhibit as smooth a rolloff as the file infection antibodies (see Fig. 28). Full detection does not occur until 256 or more scan strings are utilized. This is due in part to the larger sample sizes used in the ID tests. The IDS also exhibits a higher sensitivity to antibody numbers. 32 antibodies give a 31% average false-negative rate and 64 antibodies drops the error down to 20%; not a large effectiveness gain. But, using 32 AV antibodies gives a 65% average false-negative rate with 64 AV antibodies, resulting in a 40% rate. The AV detectors produce much wider

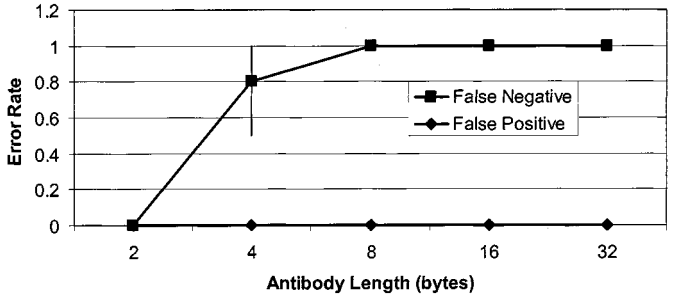


Fig. 29. Error rates versus computer-virus antibody length.

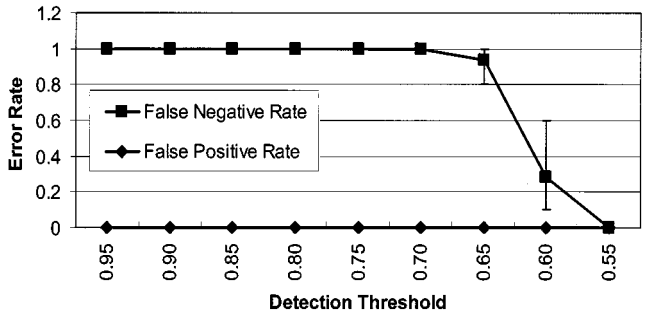


Fig. 30. Error rates versus computer-virus detection threshold.

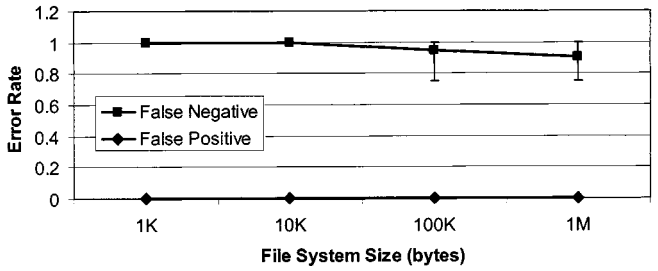


Fig. 31. Virus detection error rates versus file system size.

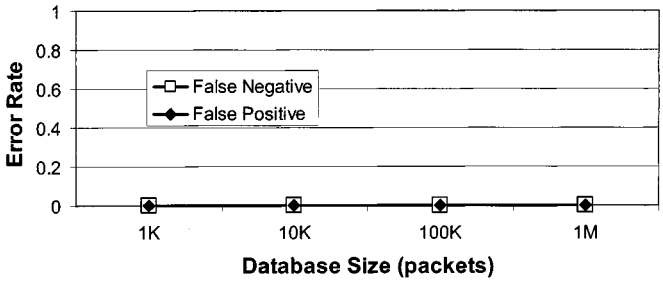


Fig. 32. ID error rates versus database size

swings in performance with changes in antibody count due to the smaller sample size.

The length of the antibody affects its specificity as a AV detector. The error rate for long detectors should be greater than the smaller more general strings. This is a by-product of a long detector's need to search an exponentially larger space in order to find a match. This trend is evident in Fig. 29. At lengths greater than 4 B, the antibodies have a complete inability to find 1 K of random nonself bytes. The extremely general, 2-B strings are able to find all nonself files with no variance. In the middle between these two extremes is the 4-B antibody. On average, these perform better than the longer strings, but they also only detect nonself 20% of the time. The variance seen with this length is indicative of its position between too general and too specific. The random generation of 4-B antibodies can place them on either side of a present or future nonself boundary. For these tests, a 0.7 detection threshold is used. The error rate graph (see Fig. 29) is a reflection of the probability of a matching value occurring about this threshold. Therefore, selecting the antibody length determines its specificity, but this must be matched with an appropriate affinity threshold in order to obtain the desired error rate. In essence, reducing the detection threshold creates a more general detector, no matter what its length.

Fig. 30 depicts the effect of the detection threshold on the false-negative rate for the computer-virus attack. For this test, each run consists of a detector with 32 8-B antibodies against randomly generated self and nonself files. The IDS was only tested against real-world data, so its results are presented in the next section.

The AV 100% false-negative rate at a 0.7 threshold matches the same result in Fig. 29. At threshold values less than 0.7, the antibody set becomes an increasingly effective nonself discriminator. An affinity threshold of 0.55 results in a 100% effective detector with no variance.

The final laboratory error rate tests examine the effects of the file system size on the error rate for file infector viruses (see Fig. 31). The results indicate only a minor influence. This is not surprising as a larger set of nonself bytes simply gives the detector more chances to encounter a match. For these tests, detectors used 16 8-B antibodies and a detection threshold of 0.7. The high threshold value should result in a 100% false-negative rate (see Fig. 30). The AV detector searched up to a 2-MB file system containing up to 1 MB of random nonself bytes. In each case, the file system was made up of one part self and one part nonself bytes. In practice, the likelihood of 1 MB of nonself appearing on an individual system is all but impossible. Because most viruses are smaller than 5 KB [48], the accumulation of 1 MB of nonself bytes would require a significant number of simultaneous infections or the addition of large infected applications. This is an event that is so remote that its occurrence is impossible without sabotage. Therefore, this example is mostly pedagogical, but the goal was to understand the effect of nonself size on the system error rates through experimentation. In a practical environment (5–10 KB of nonself), the size of nonself has no measurable effect on the false-negative error rate, except to keep it high.

Fig. 32 presents the results of developing 256 antibodies over a set of self data (e.g., 1-K packets) and then testing over a 20% larger set for the purpose of ferreting-out surprises. Increasing the amount of simulated traffic by including more of the LL's self data had minimal effect on the error rates. Only two to three false-positive errors emerged in the tests with the larger databases. This tends to imply that the self data is relatively homogeneous; further exploration using real-world data is required. The other obvious experiment, training on a small set of this self data and then testing on a significantly larger set, contributes little beyond a better feel for the homogeneity of the LL's data until affinity maturation and costimulation are included (areas currently under development).

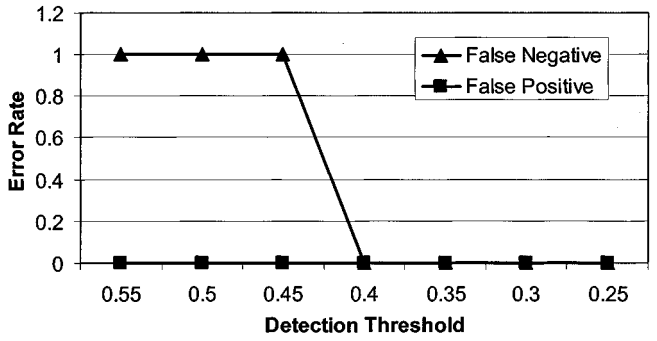


Fig. 33. Computer-virus detector's ability to detect nonself.

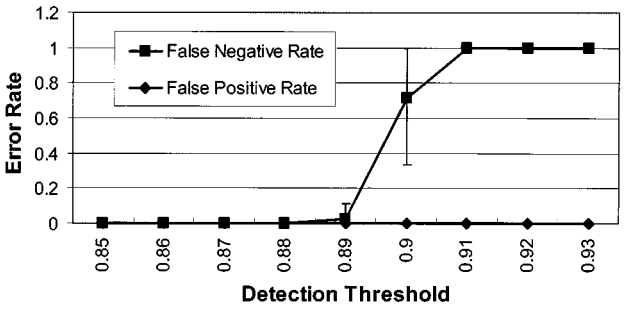


Fig. 34. Error rates versus ID threshold.

These results indicate that each detector should field as many generic antibodies as possible in order to minimize the false-negative rate. However, the use of highly generic as well as large numbers of antibodies contribute to an increased negative-selection time, but negative selection is necessary to force the false-positive rate to 0%. An engineering tradeoff must be made between negative-selection time and system effectiveness. Once the desired antibody length is selected for the virus domain (ID antibodies are 320 bits), the detection threshold must be tuned to the antibody-matching function probability density in order to create a system that actually detects nonself with the desired frequency. This tuning must also take into account the specific detector domain as the differing domains result in different sensitivities to the various parameters.

### *D. Real-World Effectiveness*

The previous tests have shown that the system operates as designed and is able to successfully detect the existence of nonself within a set of self strings. However, these results were gained by testing the system against randomly generated self and nonself bytes (for AV) or LL's training set IP packets (for ID). In order to be truly effective, the system must be able to detect actual malicious code among a larger set of known self applications or intrusion attempts within a set of valid network service request packets.

The first test against other than random bytes uses a polar input set. For this test, self is made up of all ones, while nonself consists of all zeros (see Section XI-B, Table VI). Interestingly, the randomly generated 8-B antibodies have a harder time finding this consistent nonself set (see Fig. 33) than a set of random bytes 30. Full detection only occurs with an activation threshold of 0.4 or less. The 100% error rate difference between 0.45 and 0.4 is indicative of the consistent polar nature of the self and nonself sets. Once one detector is able to bind with a string of zeros, it is able to bind with all of nonself. This results in a 0% false-negative rate once the detection threshold is crossed. This test does not provide much useful information in itself other than the dramatic effect a proper detection threshold selection can make, but by comparing with the similar data obtained using random nonself bytes (see Fig. 30), an interesting difference emerges. This test shows 100% detection at a threshold of 0.4, while with random strings, 100% detection occurs at a 0.55 threshold. Previous data indicated that the detection threshold should be tuned based on the antibody length. Additionally, this test indicates that tuning should also be done based on the contents of self and nonself, a result that is further validated by changing the domain to ID (see Fig. 34).

These data also show that antibody generation could be improved based on knowledge of existing self and nonself bytes. An *a priori* examination of nonself would have revealed that a single antibody pattern consisting of all zeros could have matched, with the highest affinity possible, all nonself in this system (an improvement would seed antibody generation with known virus, or network attack signatures). Because of the influence of search space contents on system effectiveness, tests against actual viruses and captured network intrusion attempts were conducted.

The IDS test utilizes the LL training set seeded with captured probing attacks conducted using Nessus [45], configured to simulate a complete port scan from one machine onto another, both having LL self data IP addresses. The results indicate a dramatic difference in the ID domain with respect to the effective range of threshold values compared to the file infection detector (see Fig. 34). These results are similar to the polar self/nonself test above, but much higher threshold values can be used effectively to give the system 100% detection. The results are also somewhat more sensitive to threshold value than the AV antibodies: changing the threshold by 0.03 (versus 0.05) results in a difference between 100% and 0% false-negative rate.

The real-world virus test utilizes test input five (see Section IX-B, Table VI). This test suite consists of 196 KB of application programs and 136 B in the two EICAR "viruses" (see Section XI-B). This test also reveals the system's ability to detect, as of yet, unknown viruses. For example, NAV can detect the EICAR68 test string 100% of the time, while it has a 100% false-negative rate for the newly created EICARPAU test string. Fig. 35 presents the error rates for various detection thresholds. The detection rates represent the total rate for 90 runs with a variable number of antibodies per detector. A roughly 10% false-negative error rate is the best result when using a detection threshold of 0.6 or less.

In general, the system was able to detect the new virus strain at a rate only slightly less than that of the known virus. Additionally, the system found both nonself files at a rate equal to or slightly less than the least detected strain. In these cases, the addition of affinity maturation or antibody optimization to improve the antibody false-negative rate could be highly useful. An affinity maturation capability could either evolve antibodies to recognize one virus very well or evolve a general detector that binds to both strings equally.

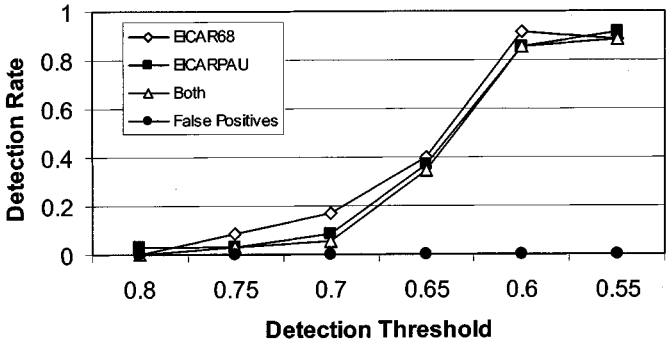


Fig. 35. Detection rate for known and unknown viruses.

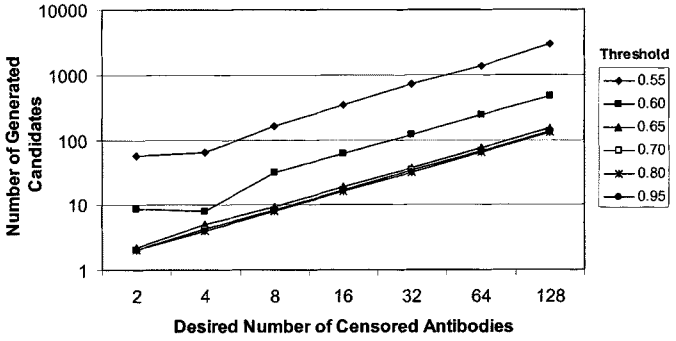


Fig. 36. Effects of matching threshold on negative-selection candidate pool size.

A tuned detection threshold results in an 89% detection rate for both strings. NAV produces a constant 50% false-negative rate. This system out performs NAV for detection thresholds of 0.6 or below. The introduction emphasizes the inability of current AV software to adapt and recognize new viruses. This system is able to detect the new strain with approximately a 9% false-negative rate. The tradeoff with the immune system methodology is the probability of detection, while current systems utilize deterministic scanning to give 100% detection of known viruses. However, through careful tuning, a 0% false-negative rate can be obtained (see Fig. 30). Beyond this, additional coverage could possibly be gained by utilizing distributed detectors that share successful antibodies. It is hypothesized that an improved error rate can be gained in this manner through a multiagent collective self-defense.

### *E. Antibody Candidate Pool Size*

The experiments on the system negative-selection time suggest that a performance increase can be gained by over generating the number of required antibodies and then censoring this large pool down to the required number. However, such an algorithm requires understanding what size the initial pool of uncensored scan strings should be. Forrest's research on the $r$-contiguous-bits algorithm validates theoretical results that the required number of initial strings $N_{R_0}$ grows with the the probability of a match, the number of final strings required, and the size of self [30]. The Rogers and Tanimoto similarity rule produces similar results.

This experiment varies the detection threshold and the number of final antibodies required against the TIMID virus-infected application-suite input 6 (see Section XI-B, Table VI). The results indicate that the size of $N_{R_0}$ increases linearly with the number of required antibodies and exponentially with a decrease in the detection threshold (see Fig. 36). The higher detection thresholds all require the approximately same number of initial candidates, with a break in this trend occurring at a threshold of 0.65. The required number of candidates increases dramatically thereafter. This phenomenon is roughly the inverse of the results seen in Figs. 30 and 35. As the detection threshold decreases, the antibodies become more general. This results in an increased number of matches on self during censoring, and improved nonself detection during employment. The small false-negative rates that are required for an effective system require the upfront investment in a large antibody candidate pool. The near 100% detection rate seen at a threshold of 0.6 requires about four times the number of immature strings as naive ones. In order to obtain a 0% false-negative rate at a 0.55 threshold, a 23:1 ratio is required. This is a reflection of the increased negative-selection time versus coverage tradeoff seen in earlier experiments (see Section XII-A). This testing has not been performed in the ID domain, but similar results are expected.

Once again, the performance of the system requires a tradeoff between coverage, speed, and memory. The selection of the system parameters, such as number of antibodies per detector and the detection threshold, can have a dramatic affect on the system efficiency and effectiveness. At a detection threshold of 0.55, generating 128 antibodies requires an initial pool of 3020 candidates on average. Previous results indicate that at least 64 4-B antibodies, at a detection threshold of 0.60 or less, is required in order to reduce the false-negative error rate to within effective limits. This requires the generation of several hundred to several thousand candidate antibodies for censor.

## XIII. Qualitative Analysis

The overall goal is to create an agent-based CDIS. This was accomplished successfully and two layers of defense have been implemented. Effective system and local models of immune system operation were constructed that realize improvements over current AV and packet-based ID solutions. Based on these models, the multilayered implementation provides an effective solution for the detection, identification, and elimination of computer viruses and network attacks. The prototype was used to gain insight into the efficiency and effectiveness of an agent-based AIS. The successful use of agents and the integration of pattern recognition principles are valuable contributions to the immunological computation community.

This research was conducted by integrating many different domains including immunology, immunological computation, malicious code, multiagent systems, and parallel and distributed computation. Because of the diverse amalgamation of ideas, conclusions are discussed from a variety of perspectives. The conclusions are based on the analysis of this design implementation.

1) *System Models:* The system and local models (see Section VIII-B) for this CDIS are created based on ideas from biology, the self-adaptive CVIS [6], the antibody

lifecycle [12], and parallel computation. The separation of tasks into a logical hierarchy supports the reduction of the computational burden by allocating responsibilities to dedicated agents operating at the appropriate level. By integrating this structure with the prevention focus of the computer health system [26], a system-wide "computational health management" infrastructure is created that emphasizes preventative measures through information sharing. It is hypothesized that such an infrastructure will allow for the early identification and elimination of wide spread attacks. It also provides a forum for a collective self-defense by enabling the sharing of successful antibodies among individual detectors in the "population." This diversity that is used to the advantage of the entire system is the result of the local model.

Each detector on each node within the system independently generates and manages its own antibody set. The computational burden on individual nodes is reduced by limiting the local number of antibodies. This distributes the cost of generation and negative selection across the system. These tasks can also be performed in parallel. Even though the detection capabilities at the local node are limited to the antibodies on hand, the full power of all the system scan strings can be utilized though information sharing via vaccinations. If placed within a broadcast LAN environment (e.g., Ethernet), then all ID agents can employ their antibodies simultaneously. Vaccinations between LAN's would support an even larger collective self-defense. Additionally, each node is continually searching the nonself space through the "programmed cell death" within the local detector string lifecycle. This realizes the greatest advantage of this system over current methodologies, which is the ability to recognize as of yet unknown viral infections. The power gained through the partitioning of tasks and the sharing of information is accomplished through distributed, collaborating agents.

2) *Agents:* The BIS is made up of many individual entities, each with their own "goals" and "services." Because of this, mapping the capabilities of these entities to software agents is an intuitive task. Additionally, the BIS components communicate through chemical signals. This can be mapped to message passing in a distributed AIS. For these reasons, the agent paradigm represents an excellent software engineering approach to AIS design.
3) *Antibodies:* The detector agents each carry a battery of several antibody scan strings. In this prototype, these are generated pseudorandomly. This provides a quick production method and because the exact locations of nonself within the search space are unknown, probably provides as good a method as any given all the possible nonself instantiations.

Testing shows that there exists an engineering tradeoff between the specificity and generality of an antibody. Short strings are more general because they reduce the dimensionality of the self/nonself space and, hence, cover a larger area. A 4-B antibody is shown to provide the coverage of a general detector string without the high negative-selection cost of being too general. Current AV solutions utilize 16-B scan strings in order to help eliminate the threat of false-positive errors. The CDIS accomplishes this through negative selection. However, short antibody lengths will not be able to adequately distinguish between self and nonself in cases where their differences are fine grained. The result is undetectable holes in the detector's ability to recognize nonself. It has been shown that eliminating holes is impossible with a single matching rule [36] so multiple approaches are required to completely cover the nonself space. This discussion alludes to a characterization of the self/nonself space, which has not been accomplished for either problem domain. Future activities in this area could lead to the improved generation of antibodies through enhancing the random search by steering the generation algorithm toward known nonself areas of the search space.

4) *Management Advantage:* Current AV and ID solutions are monolithic and provide little or no system wide management capabilities. Each desktop locally runs the complete AV package and separate network segments do not share intrusion information in real-time. All decisions for what and how to scan are left to the user/system administrator. Even the addition of signature updates, vital to the continued effectiveness of the system, are often the task of the individual user to manually integrate. This prototype CDIS eliminates these problems and provides a framework for system metric reporting.

By using autonomous agents, this CDIS all but eliminates individual user interaction. Vaccinations and infection responses are controlled and directed by the agents at the network and system levels. Additionally, current system status is passed up the chain. This allows for automated metric collection, system status evaluation, and trend analysis. With the addition of appropriate logic, system-wide infection epidemics can be recognized and eliminated in real-time.

Organizations are increasingly interested in reporting computer-security incidents. Incident reports must be compiled and passed up the management chain. With this architecture, attack incidents are already reported up the hierarchy. Automated incident report generation and statistics could be added to the metric generation duties of the controller agents without much difficulty. This has the potential to save money and manpower that are currently being used to generate, report, and collate incident reports. Also available could be a real-time status display for infection incidents across the entire system. A live system status on malicious code or network intrusion incidents could be generated and pictorially presented.

By integrating the ideas of a system hierarchy [49] with the management and oversight processes of the public health system [26], this distributed agent-based CDIS provides a superior capability for system wide management and elimination of the virus threat over current solutions.

5) *Issues:* There are several issues that remain unaddressed by this system including security and a time-varying definition of self. No security layer is implemented in this pro-

totype: the distributed nature of the system leaves it wide open to spoofing attacks that can compromise system integrity. Encrypted channels, digitally signed messages, and other technologies are required in order to ensure trusted conversations. JSDT can easily support these additions, but what is ultimately required is a quality control mechanism for critical system components.

In the current implementation, the system could be "trained" to generate an autoimmune reaction. By performing negative selection on nonself, some censored antibodies could react against self strings. These antibodies could then be passed on to other nodes using spoofed vaccination messages. The result would be false-positive detections and the possible elimination of valid self applications or the blocking of valid network accesses. A trusted quality control mechanism is needed to oversee alarm generation and the dispensing of vaccinations.

Another problem is the steadily changing definition of self. Programs are routinely added and deleted from most desktop computers, and new users and network services are also added regularly. One of the major differences between the virus detection portion of CDIS and the network ID portion is that, realistically, self is not static in networks. While it is conceivable that, in a corporate server environment, new applications are installed at a slow enough rate to assume a static file system, even fairly consistent networks tend to have traffic patterns that shift over time. It is unrealistic to expect a zero-percent false-positive rate, for antibodies perfectly-trained on today's data may falsely detect acceptable, but new traffic tomorrow. Under conditions of high network traffic, too many false-positive errors will become intolerable, essentially creating a self-induced denial-of-service attack.

Similarly, the addition of a new application may require recensoring of the antibodies. Care must be taken to ensure the new software is not already infected. An alternative is to scan with the current antibodies: a positive detection could indicate the presence of a virus or recognition that this "self" has not been encountered before by the system. The decision on whether this is a false-positive error rests with the system administrator, and is accommodated by the system through the costimulation function of the antibody lifecycle (see Section VIII-D, Fig. 11). This approach is far from perfect, since it does not provide assurance of detection and elimination, features essential to system effectiveness.

## XIV. Future Research

1) *Improved Scanning and Negative-Selection Speed:* The current system can produce naive antibodies in 1.45 years for an 8-GB drive and scan that drive in 1.05 years. This prototype system efficiency needs to be improved in order to be operationally viable.
2) *Parallel Censoring:* The prototype algorithm generates and performs negative selection sequentially. The algorithm execution time can be greatly reduced by generating an excess number of antibodies and then censoring them all in parallel. During negative selection, those antibody strings matching self are removed from the candidate population. After censoring, only naive strings remain. A sufficiently large number must be generated initially in order to ensure that enough remain after negative selection. This number of initial candidates must be estimated based on the antibody length, contents of self, detection threshold, and the number of remaining strings required after negative selection.
3) *Efficient String Matching:* Improved methods of string pattern matching could be integrated to increase the performance of the matching algorithm. A common method used in spell checking is to *a priori* construct a directed graph of the patterns. This is then used to process the input string against all patterns in a single pass by "walking" the graph [50].
4) *Antibody Creation:* The prototype uses a pseudorandom number generator to create antibody candidates. These are then censored at a very high rate to produce valid detection strings. Improved antibody generation schemes could reduce the censoring rate by directing the creation algorithm to known areas of the nonself space or seeding the initial antibody population with known attack signatures. This would improve the generation and negative-selection efficiency.
5) *Affinity Maturation* The current implementation of deploying randomly generated antibodies can result in multiple matches on the same antigen. Affinity maturation could be implemented to conserve resources by only retaining the antibody with the highest affinity. This could be extended to include hypermutation and clonal selection algorithms to create evolved copies of high affinity antibodies. This has the possibility of improving the system adaption process and also increasing the detection of related viral strains.
6) *Metrics:* One of the goals of the controller agent is to produce metrics on system performance (see Table IV). This functionality is necessary for management insight into system operation and in order to understand the system wide impact of viruses. Real time displays could also be created based on the metric information. This functionality is not currently not implemented.
7) *Additional Detectors:* Currently, only file infector viruses and packet-based network attacks are detected with the prototype system. Additional agent types need to be created in order to detect and remove the other viral threats, such as macroviruses, or to implement more complex state-based ID. A complete set of detector types is required to create a multilayered defense-in-depth.
8) *Robust Deployment:* The prototype contains very little code to deal with system failures. However, the system architecture is designed to one-day accommodate such functionality. Features should be added to support the graceful degradation of service in the face of failure, instead of system collapse. This could include backup agents, such as monitors that automatically fail over to their adjacent peers, and communications timeouts with recovery.

9) *Security:* The current system is highly vulnerable to spoofing and denial of service attacks. For instance, erroneous vaccinations could easily be sent to a detector, which could cause an autoimmune reaction. The prototype architecture easily supports the addition of security layers, such as secure socket communication and agent authentication, but they are not currently implemented. These would have to be added, especially for a wide area network deployment, in order to overcome the security problems associated with system compromise.

## XV. Conclusion

The system design integrates the power, flexibility, adaption, and capabilities of the BIS into an architecture realizable in the information system domain. Based on the models, the prototype implementation provides an effective solution for the detection, identification, and elimination of malicious code and bad packets. The level of effectiveness is tunable through the proper selection of the number of antibodies, the antibody length, and the detection threshold. These must be selected based on the contents of known self and with an understanding of their ramifications on negative-selection time, scan time, and nonself space coverage.

The use of the agent paradigm facilitates the construction of an AIS because of the performance limitations of a monolithic implementation and the biological basis for the architecture can be viewed as a system of collaborating agents [51]. While using agents improves the understanding of the system design and the mapping to the biological domain, the deployment of the agents must be done by considering the principles of parallel software design in order to improve performance. For an agent-based CDIS, this involves reducing communication and placing detection agents near their I/O sources.

This CDIS design is scaleable in terms of scope and coverage through the simple addition of new agent types and participating system nodes. The prototype implements file system and IP packet detection, but a more complete multilayered defense could be realized by adding agent types for monitoring memory, email, boot sectors, complex intrusions, and more. Additionally, because the JSDT provides lookup services, agents can join or leave the system at anytime.

At its current level of maturity, the prototype does not provide for a practical implementation nor unobtrusive operation. The Java implementation provides a good prototype environment, but its speed limits the system usability. The negative-selection and scanning times measured in years are unacceptable for a practical system. An implementation improvement to increase the system speed is paramount to future system viability.

The agent-based CDIS offers detection and management capabilities that are absent from current deployed solutions. The abilities of these facets working together promises an enterprise-wide computer-security solution. At the heart of CDIS is the ability to proactively generate antibodies capable of detecting nonself data; the research presented herein investigates a method of generating antibodies for the computer-virus and network intrusion problem domains. The preliminary results, though limited, indicate that this approach holds promise and deserves continuing investigation.

## References


[1] P. K. Harmer, "A distributed agent architecture for a computer virus immune system," M.S. thesis, Air Force Inst. Technol., Wright-Patterson AFB, OH, Mar. 2000.

[2] P. D. Williams, "Warthog: Toward an artificial immune system for detecting 'low and slow' information system attacks," M.S. thesis, Air Force Instit. Technol., Wright-Patterson AFB, OH, Mar. 2001.

[3] Symantec. (2001, Oct.) Symantec Security Response—Definitions Added. [Online]. Available: http://www.symantec.com/avcenter/defs.added.html

[4] M. Leon, "Internet virus boom," *Infoworld*, vol. 22, no. 3, pp. 36–37, Jan. 2000.

[5] D. Dasgupta, Ed., *Artificial Immune Systems and Their Applications*, Heidelberg, Germany: Springer-Verlag, 1999.

[6] G. B. Lamont, R. E. Marmelstein, and D. A. Van Veldhuizen, "A distributed architecture for a self-adaptive computer virus immune system," in *New Ideas in Optimization*. New York: McGraw-Hill, 1999, Advanced Topics in Computer Science Series, ch. 11, pp. 167–183.

[7] E. Benjamini, G. Sunshine, and S. Leskowitz, *Immunology: A Short Course*, 3rd ed. New York: Wiley, 1996.

[8] P. D. Williams, K. P. Anchor, J. L. Bebo, G. H. Gunsch, and G. B. Lamont, "CDIS: Toward a computer immune system for detecting network intrusions," in *Proc. Fourth Int. Symp. Recent Advances in Intrusion Detection*, Oct. 2001, pp. 117–133.

[9] J. O. Kephart and W. C. Arnold, "Automatic extraction of computer virus signatures," in *Proceedings of the 4th Virus Bulletin International Conference*, R. Ford, Ed. Abingdon, U.K.: Virus Bulletin Ltd., 1994, pp. 179–194.

[10] J. O. Kephart, G. B. Sorkin, M. Swimmer, and S. R. White, "Blueprint for a computer immune system," in *Proceedings of the Virus Bulletin International Conference*. Abingdon, U.K.: Virus Bulletin Ltd., 1997.

[11] S. Forrest, S. A. Hofmeyer, and A. Somayaji, "Computer immunology," *Commun. ACM*, vol. 40, no. 10, pp. 88–96, Oct. 1997.

[12] S. A. Hofmeyr and S. Forrest, "Immunity by design: An artificial immune system," in *Proceedings of the Genetic and Evolutionary Computation Conference*. San Mateo, CA: Morgan Kaufmann, July 1999, pp. 1289–1296.

[13] D. Dasgupta. Immunity-based intrusion detection systems: A general framework. presented at 22nd Nat. Information Systems Security Conf.. [Online]. Available: http://csrc.nist.gov/nissc/1999/proceedings/papers/p11.pdf

[14] ——, "An artificial immune system as a multi-agent decision support system," in *Proc. IEEE Int. Conf. Systems, Man and Cybernetics*, Oct. 1998, pp. 3816–3820.

[15] K. Mori, M. Tsukiyama, and T. Fukuda, "Multi-optimization by immune algorithm with diversity and learning," in *Proc. Second Int. Conf. Multiagent Systems*, Dec. 1996, pp. 118–123.

[16] E. Hart, P. Ross, and J. Nelson, "Producing robust schedules via an artificial immune system," *Proc. IEEE Int. Conf. Evolutionary Computing*, pp. 464–469, May 1998.

[17] Her Majesty's Office of Information. (1996, Sept.) Antibodies teach computers to learn. [Online]. Available: http://www.aber.ac.uk/~jot/ISYS/hmoi.html

[18] J. Hunt and D. Cooke, "The ISYS Project: An Introduction," Univ. Wales, Aberystwyth, Aberystwyth, U.K., Tech. Rep. IP-REP-002, 1996.

[19] S. A. DeLoach, "Multiagent systems engineering: A methodology and language for designing agent systems," in *Proc. Int. Bi-Conf. Workshop Agent-Oriented Information Systems*, May 1999, pp. 45–57.

[20] R. Skardhamar, *Virus Detection and Elimination*. New York: Academic, 1996.

[21] Cult of the Dead Cow. (1999, July) Back Orifice 2000. [Online]. Available: http://www.bo2k.com

[22] F. B. Cohen, *A Short Course on Computer Viruses*, 2nd ed. New York: Wiley, 1994.

[23] L. J. Hoffman, Ed., *Rogue Programs: Viruses, Worms, and Trojan Horses*. New York: Van Nostrand Reinhold, 1990.

[24] M. A. Ludwig, *The Little Black Book of Computer Viruses*. Show Low, AZ: American Eagle, 1996.

[25] Symantec Security Response—W32.Nimda.A@mm, Symantec. (2001, Oct.). [Online]. Available: http://www.symantec.com/avcenter/venc/data/w32.nimda.a@mm.html

[26] K. J. Cardinale and H. M. O'Donnell, "A constructive induction approach to computer immunology," M.S. thesis, Air Force Inst. Technol., Wright-Patterson AFB, OH, Mar. 1999.

[27] A compendium of NP optimization problems (1999, May). [Online]. Available: http://www.nada.kth.se/~viggo/problemlist/compendium.html

[28] S. A. Hofmeyr, "An immunological model of distributed detection and its application to computer security," Ph.D. dissertation, Univ. New Mexico, Albuquerque, NM, 1999.
[29] C. A. Janeway Jr., "How the immune system recognizes invaders," *Sci. Amer.*, vol. 269, no. 3, pp. 73–79, Sept. 1993.
[30] S. Forrest, L. Allen, A. S. Perelson, and R. Cherukuri, "Self-nonself discrimination in a computer," in *Proc. IEEE Symp. Research in Security and Privacy*, May 1994, pp. 202–212.
[31] D. Dasgupta and F. Nino, "A comparison of negative and positive selection algorithms in novel pattern detection," in *Proc. IEEE Int. Conf. Systems, Man, and Cybernetics*, vol. 1, Oct. 2000, pp. 125–130.
[32] J. T. Tou and R. C. Gonzalez, *Pattern Recognition Principles*. Reading, MA: Addison-Wesley, 1974.
[33] M. Nadler and E. P. Smith, *Pattern Recognition Engineering*. New York: Wiley, 1993.
[34] A. S. Perelson and G. Weisbuch, Eds., *Theoretical and Experimental Insights Into Immunology*. New York: Springer-Verlag, 1992, ch. Probability of self-nonself discrimination, pp. 63–70.
[35] J. O. Kephart, "A biologically inspired immune system for computers," in *Proc. Fourth Int. Workshop Synthesis and Simulation of Living Systems*, July 1994, pp. 130–139.
[36] P. D'haeseleer, "An immunological approach to change detection: Theoretical results," in *Proc. 9th IEEE Computer Security Foundations Workshop*, June 1996, pp. 18–27.
[37] A. Somayaji, S. Hofmeyer, and S. Forrest, "Principles of a computer immune system," in *Proc. New Security Paradigms*, Sept. 1997, pp. 75–82.
[38] R. E. Marmelstein, D. A. Van Veldhuizen, P. K. Harmer, and G. B. Lamont, "A white paper on modeling and analysis of computer immune systems using evolutionary algorithms," Air Force Inst. Technol., Wright-Patterson AFB, OH, Dec. 1999.
[39] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Redwood City, CA: Benjamin Cummings, 1994.
[40] S. A. DeLoach, Using agentMOM, 1999.
[41] R. Burridge, *Java Shared Data Toolkit User Guide, Version 1.5*, Sun Microsystems, Mountain View, CA, Apr. 1999.
[42] H. Crowder, R. S. Dembo, and J. M. Mulvey, "On reporting computational experiments with mathematical software," *ACM Trans. Math. Software*, vol. 5, no. 2, pp. 193–203, June 1979.
[43] P. Ducklin, "Standard anti-virus test file," Eur. Inst. Computer Anti-Virus Research, Brussels, Belgium, Aug. 1999.
[44] R. K. Cunningham, R. P. Lippmann, D. J. Fried, S. L. Garfinkel, I. Graf, K. R. Kendall, S. E. Webster, D. Wyschogrod, and M. A. Zissman. Evaluating intrusion detection systems without attacking your friends: the 1998 DARPA intrusion detection evaluation. presented at Third Conf. and Workshop on Intrusion Detection and Response. [Online]. Available: http://www.ll.mit.edu/IST/ideval/pubs/1999/Evaluating_IDs_DARPA_1998.pdf
[45] Nessus Ver. 1.0.5 (2000). [Online]. Available: www.nessus.org
[46] R. Jain, *The Art of Computer Systems Performance Analysis*. New York: Wiley, 1991.
[47] R. S. Barr, B. L. Golden, J. P. Kelly, M. G. C. Resende, and W. R. Stewart, "Designing and reporting on computational experiments with heuristic methods," *J. Heuristics*, vol. 1, no. 1, pp. 9–32, Mar. 1996.
[48] Tally's Virii Link Reference. (1999, Nov.) Tally's virus collection statistics. [Online]. Available: http://www.virusexchange.org/tally/stats1.html
[49] R. E. Marmelstein, D. A. Van Veldhuizen, and G. B. Lamont, "A distributed architecture for an adaptive computer virus immune system," in *Proc. IEEE Int. Conf. Systems, Man, and Cybernetics*, vol. 4, Oct. 1998, pp. 3838–3843.
[50] A. V. Aho and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," *Commun. ACM*, vol. 18, no. 6, pp. 333–340, June 1975.
[51] K. P. Sycara, "Multiagent systems," *AI Mag.*, vol. 19, no. 2, pp. 79–92, 1998.

**Paul K. Harmer** received the B.S.E.E. degree from the California State University, Long Beach, in 1996 and the M.S.E.E. degree from the Air Force Institute of Technology, Wright-Patterson AFB, OH, in 2000.

He is currently the Director of Technical Services of the new high-performance computing distributed center for the Air Force Research Laboratory, Sensors Directorate, Wright-Patterson AFB. He is also the Program Manager for the Virtual Distributed Laboratory, which allows geographically distributed researchers to collaborate and share code, data, and programmatic information online.

**Paul D. Williams** received the B.S. degree in computer science from the University of Washington, Seattle, in 1996, and the M.S. degree in computer science from the Air Force Institute of Technology (AFIT), Wright-Patterson AFB, OH, in 2001.

His course of study at AFIT centered primarily on information operations, with significant course work in the areas of artificial intelligence and advanced algorithm design. He is currently with the Air Intelligence Agency, Lackland AFB, TX.

**Gregg H. Gunsch** received the B.S.E.E. degree from the University of North Dakota, Grand Forks, in 1979, the M.S.E.E. degree from the Air Force Institute of Technology (AFIT), Wright-Patterson AFB, OH, in 1983, and the Ph.D. degree in electrical engineering from the University of Illinois at Urbana-Champaign, Urbana, in 1991.

He has over 15 years of experience in developing synergistic computer-human systems through the application of artificial intelligence techniques. He is currently responsible for the information systems security/assurance (information warfare) curriculum at AFIT.

**Gary B. Lamont** received the B.S. degree in physics and the M.S.E.E. and Ph.D. degrees from the University of Minnesota, Minneapolis, in 1961, 1967, and 1970, respectively.

He is currently a Professor of Electrical and Computer Engineering at the Air Force Institute of Technology, Wright-Patterson AFB, OH, where he directs the parallel and distributed computing and the evolutionary computation research groups. Previously, he was an Engineering Systems Analyst for the Honeywell Corporation for six years. He has authored or coauthored a book, several book chapters, and over 100 papers. His current research interests include parallel/distributed computation, evolutionary computation (genetic algorithms, evolutionary strategies), combinatorial optimization problems (single objective, multiobjective), formal methods, software engineering, digital signal processing, intelligent and distributed control systems, computational and numerical methods, and computer-aided design.